2024/10/15 Tuesday 贪心和矩阵

Updated 0425 GMT+8 Oct 15, 2023

2024 fall, Complied by Hongfei Yan

> Log:
>
> 2024/10/15 复制自 20231017_notes.md，根据本学期进度进行修改。

2024年10月份，主要是掌握矩阵、贪心题目，最好能涉及到简单的DP题目。

时间复杂度在上一讲(2024108)，但是上次课时不够，没有展开讲，讲义里面有详细内容，重点是理解O(1)，O(n)，O(logn)。

一、Recap

1.1 OCT 月考暨选做题目满百*

感觉这部电影的重点在于刻画图灵富有传奇色彩、跌宕起伏的一生，但是专业性并不强。我之前对相关知识一无所知。在第一遍完整看完电影之后，并没能够搞清楚恩尼格玛（Enigma）密码究竟是怎么被破译的。二刷结合上网查资料之后才算有了一个大致的了解：

1. 丹尼斯顿（Dennisiton）中校指出：恩尼格玛密码机自身是一个加密工具，需要了解“设置”（Settings）才能破译。

2. 克拉克（Clarke）在和图灵（Turing）第一次聚会（也即是“故意让他喜欢你”那次）上提出“字母不可以对应自身”（No letter can be encoded as itself）（这个翻译太烂了，完全没有表达出核心意思，即，没有一个字母加密后保持不变）。

3. 发现“六点的消息中一定有三个单词”（That's three words we know will be in every 6:00 message）“weather”“heil”“hitler”。

电影中图灵所做的事情其实是拿着 3 中三个词的原文（可能只有“weather”和“hitler”）结合 2 中的规律去对密文中相同长度的连续字符串逐一对照，并对剩下的可能进行一一排查，大大减少了工作量。这其中，亚历山大（Alexander）提出的“按插板矩阵的对角线安置电线”（Run the wires across the plugboard matrix diagonally）也起到了很大作用。

电影中还有一些有意思的细节：

1. 影片开头的氰化物与中间图灵送同事苹果和结尾服下含氰化物的苹果相呼应。

2. 图灵在和警官对话时讲到了著名的“图灵测试”（虽然他没有提出这个词）在对话的最后图灵让警官判断他是机器还是人（Am I a machine?Am I a person?）警官说他不知道。但事实上，在开头，警官问图灵：“二战期间你在做什么？”（What did you do during the war?）图灵回答：“我在一个无线电公司工作。”（I worked in a radio factory.）警官再问：“你在战争时期到底在干什么？”（What did you really do during the war?）图灵便将他的经历娓娓道来。根据这个我们便可判断图灵大概率是人，因为即使到现在普通的机器面对这两个问题也理应输出同样的答案。

影片还能引起我们对于一些其他问题的思考（比如平权问题，电车难题等

上周二2024/10/08讲到了ASCII，当时也提到了chr, ord，本意就是考这个字符加密程序可能用到。

> 加密技术的历史悠久，从古代简单的密码术到现代复杂的加密算法，经历了多个阶段的发展。下面简要概述几个主要的加密方法及其历史演变：
>
> **古代加密方法**
>
> 1. **凯撒密码（Caesar Cipher）**
>   - 这是一种非常古老的替换式密码，最早可以追溯到古罗马时期。它的工作原理是将字母表中的每个字母向前或向后移动固定数量的位置。例如，使用移位3，A变成D，B变成E等。这种加密方法非常简单，容易破解。
> 2. **置换密码（Substitution Cipher）**
>   - 置换密码是通过将明文中的字符替换成其他字符来加密信息的一种方法。每个字符都可以被任意其他字符替代，但替代规则在整个消息中保持一致。尽管比凯撒密码复杂，但仍然可以通过频率分析等手段破解。
> 3. **多表密码（Polyalphabetic Cipher）**
>   - 多表密码是置换密码的一种改进，它使用多个替换表来加密文本。最著名的例子是维吉尼亚密码（Vigenère cipher），它利用一个关键词来确定使用哪个替换表。这种方法提高了安全性，因为它使得频率分析变得更加困难。
>
> **近代加密方法**
>
> 1. **机械加密机（如恩尼格玛机 Enigma）**
>   - 第二次世界大战期间，德国军队广泛使用了恩尼格玛机进行军事通信的加密。这种机器通过一系列可旋转的轮子对字母进行复杂的转换，大大增加了破解难度。不过，盟军最终还是通过数学分析和早期的计算机技术成功破译了恩尼格玛密码。
>
> **现代加密方法**
>
> 1. **对称加密（Symmetric Encryption）**
>   - 在对称加密中，加密和解密使用同一个密钥。常见的对称加密算法包括DES（Data Encryption Standard）、3DES、AES（Advanced Encryption Standard）等。AES是目前最常用的标准之一，因其高效性和安全性而被广泛采用。
> 2. **非对称加密（Asymmetric Encryption）**
>   - 非对称加密使用一对密钥：一个公钥用于加密，另一个私钥用于解密。这种方式解决了密钥分发的安全问题，因为公钥可以公开而私钥则需保密。RSA（Rivest-Shamir-Adleman）是最著名的非对称加密算法之一。
> 3. **哈希函数（Hash Functions）**
>   - 哈希函数是一种单向函数，它接受任意长度的数据作为输入，产生固定长度的输出（称为哈希值或摘要）。哈希函数主要用于数据完整性的验证，常见的哈希算法有MD5、SHA-1、SHA-256等。
> 4. **混合加密系统（Hybrid Cryptosystems）**
>   - 实际应用中，往往结合对称加密和非对称加密的优点，形成混合加密系统。例如，使用非对称加密来安全地交换对称加密所需的会话密钥，然后使用对称加密来进行大量数据的快速加密和解密。
>
> 以上只是加密技术发展史上的几个重要里程碑。随着技术的进步，加密算法也在不断进化，以应对新的安全挑战。

28674:《黑神话：悟空》之加密

http://cs101.openjudge.cn/practice/28674/

2024年8 月 20 日，广大玩家期待已久的中国首款“3A游戏”《黑神话：悟空》如期发售。游戏内容改编自中国四大名著之一的《西游记》，在正式发布前，游戏已获得业界媒体与评论家们的普遍好评，称赞其在战斗系统、视觉设计以及世界观方面的构建。游戏上线后迅速登顶多个平台的销量榜首，两周内的全球销量超过1800万份，成为有史以来销售速度最快的游戏之一。

你的朋友小xu注意到《黑神话：悟空》至今没有盗版，她感到很好奇，在网上查询后得知《黑神话：悟空》采用“D加密”技术来为防盗版提供技术支持。D加密，全称 Denuvo Anti-Tamper，是由奥地利 Denuvo 公司推出的一种防篡改技术。Denuvo Anti-Tamper 的目标是保护游戏免受盗版和破解攻击。它的原理是在游戏程序中嵌入一些特殊的代码，这些代码会进行复杂的加密和解密操作，使得黑客无法轻易地破解游戏并且在游戏中进行修改。

小xu了解后很感兴趣，于是她开始学习了加密解密相关的技术。某一天，她猜测某一段密文只是采用了一种非常简单的加密方法完成加密：每个字母对应的密文是其在字母表中的后 k 个字母，但 **'a' 被视为 'z' 的下一个字母，从而整个字母表形成一个环**。例如，如果 k=3，那么字母 'a' 将被加密为 'd'，'z' 加密为 'c'，依此类推。

现在，请你帮助她按照她的猜测完成对密文的破译（即根据密文得出其对应的明文）。

输入

第一行为一个整数 k，表示加密方法中的偏移量。(1 ≤ k ≤ 108000)
第二行为一个由字母组成的字符串 s，表示需要解密的文本。(1 ≤ |s| ≤ 342)

输出

密文对应的明文。

样例输入

```
sample1 input:
5
LfrjXhnjshj

sample1 output:
GameScience
```

样例输出

```
sample2 input:
33
IshjrTfaoDbrvun

sample2 output:
BlackMythWukong
```

提示

tags: implementation, strings

来源

2024 TA-lxy

```python
def decrypt_caesar_cipher(k, s):
    decrypted_text = []

    for char in s:
        if 'a' <= char <= 'z':
            decrypted_char = chr((ord(char) - ord('a') - k) % 26 + ord('a'))
        elif 'A' <= char <= 'Z':
            decrypted_char = chr((ord(char) - ord('A') - k) % 26 + ord('A'))
        else:
            decrypted_char = char  # Non-alphabetic characters remain unchanged
        decrypted_text.append(decrypted_char)

    return ''.join(decrypted_text)

# Sample input
k = int(input())
s = input()

# Decrypt and print the result
print(decrypt_caesar_cipher(k, s))
```

28700: 罗马数字与整数的转换

http://cs101.openjudge.cn/practice/28700/

罗马数字包含以下七种字符：I、V、X、L、C、D和M。 字符 数值 I 1 V 5 X 10 L 50 C 100 D 500 M 1000 例如， 整数2写做II，即为两个并列的I。12写做XII，即为X+II。 27写做XXVII, 即为 XX+V+II。

通常情况下，罗马数字中小的数字在大的数字的右边。但也存在特例，例如4不写做IIII，而是IV。I在V的左边，所表示的数等于大数5减小数1得到的数值4。同样地，数字9表示为IX。这个特殊的规则只适用于以下六种情况： I 可以放在 V (5) 和 X (10) 的左边，来表示4和9。 X 可以放在 L (50) 和 C (100) 的左边，来表示40和90。 C 可以放在 D (500) 和 M (1000) 的左边，来表示400和900。

题目要求：实现罗马数字和整数的转换。 1）如果输入是一个罗马数字，将其转换成整数； 2）如果输入是一个整数，将其转换为罗马数字。 无论是何种输入，整数在1到3999的范围内。

输入输出示例：

输入：III

输出：3

输入：4

输出：IV

输入：IX

输出：9

输入：58

输出：LVIII

输入：MCMXCIV

输出：1994

（M = 1000, CM = 900, XC = 90, IV = 4）

输入

一行字符（可能是罗马数字，也可能是正整数）

输出

如果输入是罗马数字，输出是正整数；
如果输入是正整数，输出是罗马数字。

样例输入

```
MCMXCIV
```

样例输出

```
1994
```

提示

tags: implementation
整数是正整数，数值范围1~3999。罗马数字中的字符是大写的。
可采用第一个字符判断输入是整数（0_{9}）还是罗马数字（不是0^{9}）。

来源

2024 TA-xjk

```python
# 定义罗马数字和整数的映射关系
roman_to_int_map = {
    'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000
}

# 定义整数到罗马数字的映射列表（从大到小顺序）
int_to_roman_map = [
    (1000, 'M'), (900, 'CM'), (500, 'D'), (400, 'CD'),
    (100, 'C'), (90, 'XC'), (50, 'L'), (40, 'XL'),
    (10, 'X'), (9, 'IX'), (5, 'V'), (4, 'IV'), (1, 'I')
```

```python
]


# 罗马数字转整数
def roman_to_int(s):
    total = 0
    prev_value = 0
    for char in s:
        value = roman_to_int_map[char]
        if value > prev_value:
            total += value - 2 * prev_value  # 处理特殊情况，如IV, IX
        else:
            total += value
        prev_value = value
    return total


# 整数转罗马数字
def int_to_roman(num):
    result = []
    for value, symbol in int_to_roman_map:
        while num >= value:
            result.append(symbol)
            num -= value
    return ''.join(result)


# 主函数，判断输入是罗马数字还是整数
def main():
    # 输入处理
    input_data = input().strip()

    # 判断输入是整数还是罗马数字
    if input_data.isdigit():
        # 输入是整数
        num = int(input_data)
        print(int_to_roman(num))
    else:
        # 输入是罗马数字
        print(roman_to_int(input_data))


# 调用主函数
if __name__ == "__main__":
    main()
```

*25353: 排队（选做）

Greedy, http://cs101.openjudge.cn/practice/25353/

有 N 名同学从左到右排成一排，第 i 名同学的身高为 hi。现在张老师想改变排队的顺序，他能进行任意多次（包括0次）如下操作：

- 如果两名同学相邻，并且他们的身高之差不超过 D，那么老师就能交换他俩的顺序。

请你帮张老师算一算，通过以上操作，字典序最小的所有同学（从左到右）身高序列是什么?

输入

第一行包含两个正整数 $N, D(1 \leq N \leq 10^5, 1 \leq D \leq 10^9)$。
接下去 N 行，每行一个正整数 hi (1<=hi<=109) 表示从左到右每名同学的身高。

输出

输出 N 行，第 i 行表示答案中第 i 名同学的身高。

样例输入

```
5 3
7
7
3
6
2
```

样例输出

```
6
7
7
2
3
```

提示

【样例解释】
一种交换位置的过程如下：
`7 7 3 6 2–> 7 7 6 3 2–> 7 7 6 2 3–> 7 6 7 2 3–> 6 7 7 2 3`

【数据范围和约定】
对于 10% 的数据，满足 N≤100；
对于另外 20% 的数据，满足 N≤5000；
对于全部数据，满足 $1 \leq N \leq 10_5, 1 \leq D \leq 10^9, 1 \leq h_i \leq 10^9$。

这俩题目，有类似的地方，排队更难。OJ25353 排队，约束窗口是高低/上下限；OJ19757 Saruman's Army，是左右限。

2023年选北京大学计算概论B课程学生 郭绍阳 同学题解。

https://www.cnblogs.com/guoshaoyang/p/17824372.html

> 初步分析

这是一个最优化问题，需要找到一个贪心准则，然后用其他算法辅助实现。一般为思路是，先找到一个贪心构造最优解的方法，先不管这个方法的复杂度，然后再寻找一些性质，进行优化。

任务：给定一个序列，通过交换元素的位置，找到其字典序最小的一个排列。

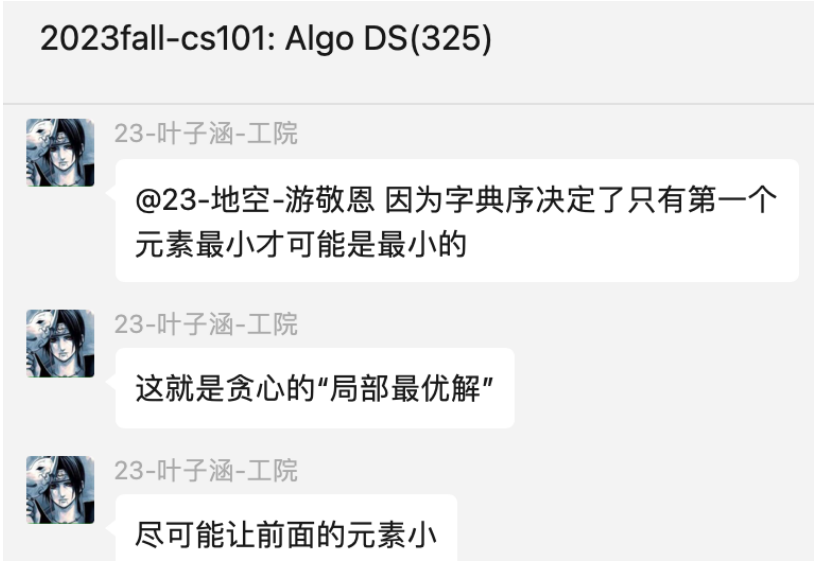
约束：任意身高差大于D的元素不对易（不对易的元素的先后关系永远不能改变后面）。

也就是每个元素前面都有一些无法翻越的“大山”，这个元素必须排到“大山”的后面。前面没有“大山”的元素，我们称为“自由节点”。
一个显然的算法就是：每次找出所有的”自由节点“，找出其中最小的一个，排在然后把它去掉。假如有多个相同大小的元素，它们必然会被连续地输出掉，所以先后顺序无影响。这个算法本身证明了其正确性(is self-proved)。

每次找出”自由节点“的代价为O(n)，只需要维护前缀最大值、最小值(最可能成为”大山“)，就可以判断当前点是否自由了，判据为 maxv - h[i] <= D and h[i] - minv<= D。时间复杂度为 O(n^2)。代码如下

```
INF = int(1e9 + 7)
N, D = map(int, input().split())
h = [int(input()) for _ in range(N)]
used = [0] * N

for _ in range(N):
minv, maxv = INF, -INF
idx, val = 0, INF
for i in range(N):
  if used[i]:
      continue
  minv = min(minv, h[i])
  maxv = max(maxv, h[i])
  if (h[i] + D >= maxv and h[i] - D <= minv and h[i] < val):
      val = h[i]
      idx = i
used[idx] = 1
print(h[idx])

```

贪心法。从最左侧的输入高度找起，按照约束条件都找出来，加入暂存列表、排序、同时标志改为True。循环找接下来还没有入选的。



402ms

```python
#  苏王捷 2300011075
N, D = map(int, input().split())
height = [0]*N
check = [False]*N
for i in range(N):
    height[i] = int(input())

height_new = []
while False in check:
    i, l = 0, len(height)
    buffer = []
    while i < l:
        if check[i]:
            i += 1
            continue
        if len(buffer) == 0:
            buffer.append(height[i])
            maxh = height[i]
            minh = height[i]
            check[i] = True
            continue

        maxh = max(height[i], maxh)
        minh = min(height[i], minh)
        if maxh-height[i] <= D and height[i]-minh <= D:
            buffer.append(height[i])
            check[i] = True
        i += 1
    buffer.sort()
    height_new.extend(buffer)

print(*height_new, sep='\n')
```

*27093: 排队又来了（选做）

http://cs101.openjudge.cn/practice/27093/

题面含义与 25353: 排队 一致，但是提供了更大的测试数据，O(n^2)会超时。

25353:排队

http://cs101.openjudge.cn/practice/25353/

内存 31800KB，时间1086ms

```
# OJ25353 排队，23-数院-胡睿诚
# 树状数组数组下标从1开始，便于理解low_bit
from collections import defaultdict
import bisect

N,D = map(int,input().split())
#N, D = 5, 3
*info, = map(int, input().split())
#info = [7, 7, 3, 6, 2]

# 内存开不了1e9那么大，使用“离散化”技巧。
# 离散化，把无限空间中有限的个体映射到有限的空间中去，以此提高算法的时空效率。
# 通俗的说，离散化是在不改变数据相对大小的条件下，对数据进行相应的缩小。

# 注意这里用set是因为我们每次只存储当前高度所对应（可能多个）点的层数最大值
heights = sorted(list(set(info)))

ass = {}
for i, h in enumerate(heights, 1):
    ass[h] = i  # 只是为了方便找到高度h排第几个

l = len(heights)
tree_l = [-1] * (l+1)   # 树状数组下标是1开始有效
tree_r = [-1] * (l+1)
#两个树状数组，分别记录从小到大和从大到小第i高的高度点所对应的层数
#这里用了变形的树状数组，不是来处理区间和而是来处理区间最大值
#这种树状数组的有效性依赖于每次修改都会把数改大，否则修改操作需要(logn)^2复杂度
ans = defaultdict(list)  # 存储分层结果：每层有哪些高度

# 树状数组辅助函数
def low_bit(x):
    return x & (-x)

def update(i, v, tree):
    while i <= l:
        tree[i] = max(v, tree[i])
        i += low_bit(i)
```

```
38
39  def get_max(i, tree):
40      res = -1
41      while i > 0:
42          res = max(res, tree[i])
43          i -= low_bit(i)
44      return res
45
46
47  for h in info:  # 按照输入的顺序（即队伍顺序）扫描
48      index = ass[h]
49      left = bisect.bisect_right(heights, h-D-1) #left下标是0开始计算
50      right = l - bisect.bisect_left(heights, h+D+1)
51      storey = 1 + max(get_max(left, tree_l), get_max(right, tree_r))
52      #递推关系。分别找到小于h-D与大于h+D的高度所对应层数的最大值
53      update(index, storey, tree_l)
54      update(l+1-index, storey, tree_r)
55      #更新高度h对应的点的层数最大值
56      ans[storey].append(h)  # 加入结果中
57
58  res = []
59  for storey in sorted(ans.keys()):
60      res.extend(sorted(ans[storey]))
61  print(' '.join(map(str, res)))
```

这份代码使用了树状数组（Fenwick Tree）来维护每个身高的“层数”，并通过离散化技巧来处理大范围的身高值。下面是代码的时间复杂度分析和详细解释。

时间复杂度分析

1. **离散化**：
   - 对身高进行排序：`O(N log N)`
   - 构建离散化映射：`O(N)`
2. **树状数组操作**：
   - 每次更新操作：`O(log N)`
   - 每次查询操作：`O(log N)`
3. **主循环**：
   - 遍历每个身高：`O(N)`
   - 对每个身高进行两次查询和两次更新：`O(log N)`

综上所述，总的时间复杂度为：`O(N log N)`

代码详细解释

1. **读取输入**：

```
N, D = map(int, input().split())
*info, = map(int, input().split())
```

2. **离散化**：
   - 将身高去重并排序，构建离散化映射。

```
heights = sorted(list(set(info)))
ass = {}
for i, h in enumerate(heights, 1):
    ass[h] = i
```

3. **初始化树状数组**：
   - `tree_l` 和 `tree_r` 分别记录从小到大和从大到小第 `i` 高的高度点所对应的层数。

```
l = len(heights)
tree_l = [-1] * (l + 1)
tree_r = [-1] * (l + 1)
ans = defaultdict(list)
```

4. **树状数组辅助函数**：
   - `low_bit`：计算最低有效位。
   - `update`：更新树状数组中的值。
   - `get_max`：查询树状数组中的最大值。

```
def low_bit(x):
    return x & (-x)

def update(i, v, tree):
    while i <= l:
        tree[i] = max(v, tree[i])
        i += low_bit(i)

def get_max(i, tree):
    res = -1
    while i > 0:
        res = max(res, tree[i])
        i -= low_bit(i)
    return res
```

5. **主循环**：
   - 遍历每个身高，计算其层数并更新树状数组。

```
for h in info:
    index = ass[h]
    left = bisect.bisect_right(heights, h - D - 1)
    right = l - bisect.bisect_left(heights, h + D + 1)
    storey = 1 + max(get_max(left, tree_l), get_max(right, tree_r))
    update(index, storey, tree_l)
    update(l + 1 - index, storey, tree_r)
    ans[storey].append(h)
```

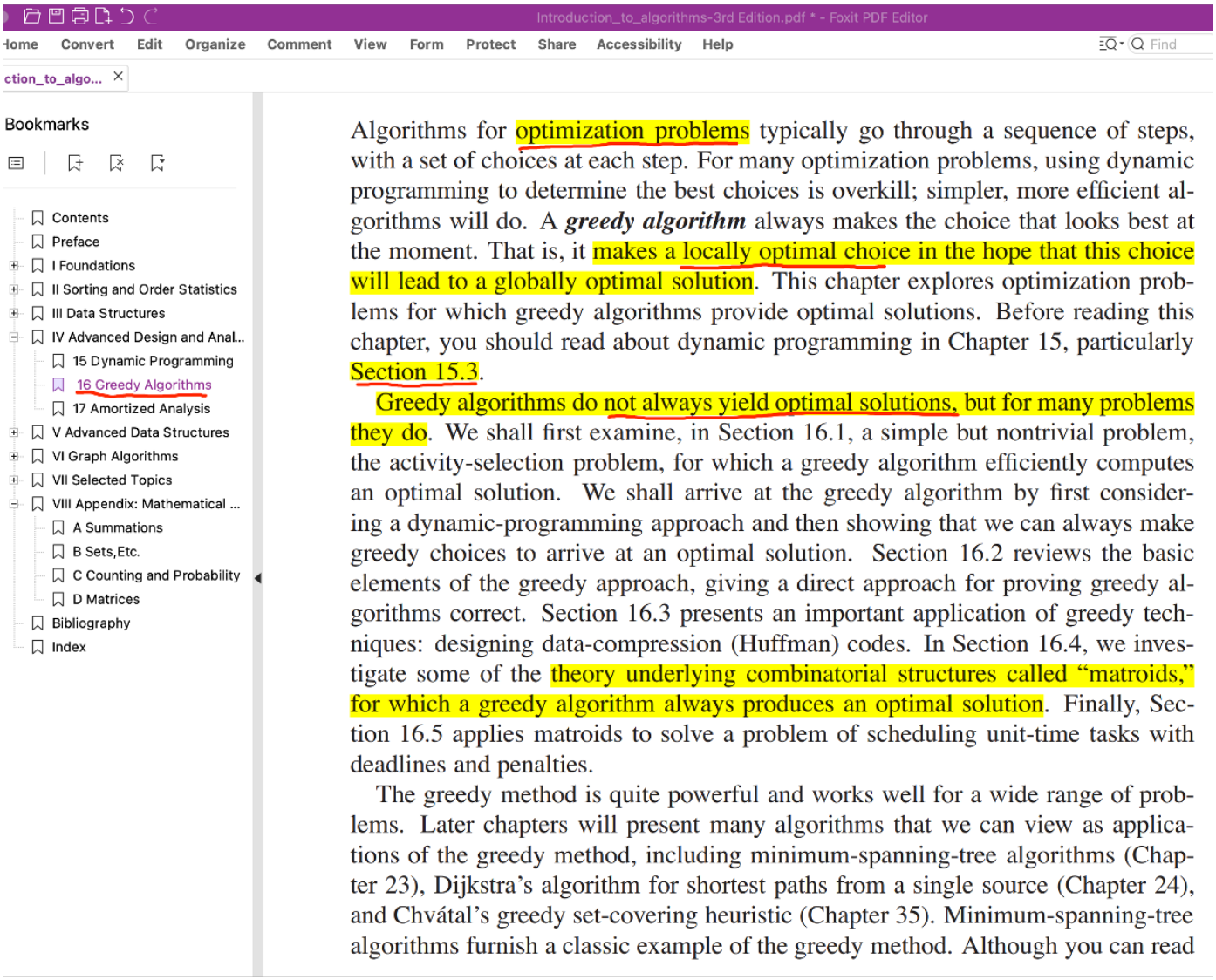
6. **输出结果**：
    - 按层排序并输出结果。

```
res = []
for storey in sorted(ans.keys()):
    res.extend(sorted(ans[storey]))
print(' '.join(map(str, res)))
```

三、Greedy

贪心算法是用来解决一类最优化问题，并希望由局部最优策略来推得全局最优结果。贪心法适用的问题一定满足最优子结构性质，即一个问题的最优解可以通过其子问题的最优解来构建。

严谨使用贪心法来求解最优问题需要对采取的策略进行证明。一般思路是使用反证法及数学归纳法，即假设策略不能导致最优解，然后通过一系列推导得到矛盾，以此证明策略是最优的，最后用数学归纳法保证全局最优。证明往往比贪心本身更难，因此一般来说，如果想到某个似乎可行的策略，并且自己无法举出反例，那么就编码实现尝试。



3.1 Sortings

排序可以按照greedy来理解，因为都有不同的优化策略。

Pyton十大排序算法源码，https://github.com/GMyhf/2024spring-cs201/blob/main/code/ten_sort_algorithms.md

Generative AI is experimental. Info quality may vary.

Yes, bubble sort is considered a greedy algorithm. A greedy algorithm makes the best possible choice at each step in the hope of finding a global optimum.

Bubble sort is a simple sorting algorithm that compares each pair of adjacent elements. It optimizes the sortedness of adjacent pairs of elements. It needs to iterate over the list several times to reach a global optimum.

Other sorting algorithms that are considered greedy include: Selection sort, Insertion sort.

Other examples of greedy algorithms include:

- Kruskal's algorithm
- Prim's algorithm
- The algorithm for finding optimum Huffman trees

> Prim算法和Kruskal算法主要用于解决无向图中的最小生成树（Minimum Spanning Tree, MST）问题。最小生成树是指在一个加权无向图中找到一棵包含所有顶点的树，且这棵树的所有边的权重之和最小。
>
> - **Prim算法**：从任意一个顶点开始构建最小生成树，逐步将距离当前树最近的一个顶点加入到树中，直到所有顶点都被包含进来。该算法适用于边数较多的稠密图。
> - **Kruskal算法**：首先将所有的边按照权重从小到大排序，然后依次选取权重最小的边，只要这条边不会与已选择的边构成回路，就将其加入到最小生成树中，直到选择了n-1条边（n为顶点数）。此算法对稀疏图较为适用。
>
> 这两种算法都能有效地找出无向图的最小生成树，但在处理有向图时则需要转换成其他形式的问题或者使用不同的算法来求解。

https://stackoverflow.com/questions/47238823/why-selection-sort-is-not-greedy

9

A selection sort could indeed be described as a greedy algorithm, in the sense that it:

- tries to choose an output (a permutation of its inputs) that optimizes a certain measure ("sortedness", which could be measured in various ways, e.g. by number of inversions), and
- does so by breaking the task into smaller subproblems (for selection sort, finding the *k*-th element in the output permutation) and picking the locally optimal solution to each subproblem.

As it happens, the same description could be applied to most other sorting algorithms, as well — the only real difference is the choice of subproblems. For example:

- insertion sort locally optimizes the sortedness of the permutation of *k* first input elements;
- bubble sort optimizes the sortedness of adjacent pairs of elements; it needs to iterate over the list several times to reach a global optimum, but this still falls within the broad definition of a greedy algorithm;
- merge sort optimizes the sortedness of exponentially growing subsequences of the input sequence;
- quicksort recursively divides its input into subsequences on either side of an arbitrarily chosen pivot, optimizing the division to maximize sortedness at each stage.

Indeed, off the top of my head, I can't think of any practical sorting algorithm that *wouldn't* be greedy in this sense. (Bogosort isn't, but can hardly be called practical.) Furthermore, formulating these sorting algorithms as greedy optimization problems like this rather obscures the details that actually matter in practice when comparing sorting algorithms.

Thus, I'd say that characterizing selection sort, or any other sorting algorithm, as greedy is technically valid but practically useless, since such classification provides no real useful information.

Share Improve this answer Follow

answered Nov 11, 2017 at 15:02

Ilmari Karonen
49.3k ● 9 ● 93 ● 153

3.2 编程题目

在问题求解时，总是做出在当前看来是最好的选择，不从整体最优上考虑。贪心算法没有固定的算法框架，关键是贪心策略的选择，贪心策略使用的前提是局部最优能导致全局最优。

双指针和二分查找是贪心算法中常用的技巧。

常规贪心题目，例如：

OJ01017：装箱问题 http://cs101.openjudge.cn/practice/01017

CF1000B: Light It Up https://codeforces.com/problemset/problem/1000/B

CF1221A: 2048 Game http://codeforces.com/problemset/problem/1221/A

01017: 装箱问题

greedy, http://cs101.openjudge.cn/practice/01017

一个工厂制造的产品形状都是长方体，它们的高度都是h，长和宽都相等，一共有六个型号，他们的长宽分别为1*1, 2*2, 3*3, 4*4, 5*5, 6*6。这些产品通常使用一个 6*6*h 的长方体包裹包装然后邮寄给客户。因为邮费很贵，所以工厂要想方设法的减小每个订单运送时的包裹数量。他们很需要有一个好的程序帮他们解决这个问题从而节省费用。现在这个程序由你来设计。

输入：输入文件包括几行，每一行代表一个订单。每个订单里的一行包括六个整数，中间用空格隔开，分别为*1*至*6*6这六种产品的数量。输入文件将以6个0组成的一行结尾。

输出：除了输入的最后一行6个0以外，输入文件里每一行对应着输出文件的一行，每一行输出一个整数代表对应的订单所需的最小包裹数。

解题思路：4*4, 5*5, 6*6这三种的处理方式较简单，就是每一个箱子至多只能有其中1个，根据他们的数量添加箱子，再用2*2和1*1填补。1*1, 2*2, 3*3这些就需要额外分情况讨论，若有剩余的3*3,每4个3*3可以填满一个箱子，剩下的3*3用2*2和1*1填补装箱。剩余的2*2，每9个可以填满一个箱子，剩下的与1*1一起装箱。最后每36个1*1可以填满一个箱子，剩下的为一箱子。

样例输入

```
0 0 4 0 0 1
7 5 1 0 0 0
0 0 0 0 0 0
```

样例输出

```
2
1
```

来源：Central Europe 1996

直接用总数把bcdef占的位置都减掉就可以了，思路就清晰起来了。**运用列表，避免多个 if else。

```
import math
rest = [0,5,3,1]

while True:
    a,b,c,d,e,f = map(int,input().split())
    if a + b + c + d + e + f == 0:
        break
    boxes = d + e + f           #装4*4, 5*5, 6*6
    boxes += math.ceil(c/4)     #填3*3
    spaceforb = 5*d + rest[c%4] #能和4*4 3*3 一起放的2*2
    if b > spaceforb:
```

```
        boxes += math.ceil((b - spaceforb)/9)
    spacefora = boxes*36 - (36*f + 25*e + 16*d + 9*c + 4*b)      #和其他箱子一起的填的1*1

    if a > spacefora:
        boxes += math.ceil((a - spacefora)/36)
    print(boxes)
```

12559: 最大最小整数 v0.3

greedy/strings/sortings, http://cs101.openjudge.cn/practice/12559

假设有n个正整数，将它们连成一片，将会组成一个新的大整数。现需要求出，能组成的最大最小整数。

比如，有4个正整数，23，9，182，79，连成的最大整数是97923182，最小的整数是18223799。

输入

第一行包含一个整数n，1<=n<=1000。
第二行包含n个正整数，相邻正整数间以空格隔开。

输出

输出为一行，为这n个正整数能组成的最大的多位整数和最小的多位整数，中间用空格隔开。

样例输入

```
Sample1 in:
4
23 9 182 79

Sample1 out:
97923182 18223799
```

样例输出

```
Sample2 in:
2
11 113

Sample2 out:
11311 11113
```

提示

位数不同但前几位相同的时候。例如： 898 8987，大整数是898+8987，而不是8987+898。

来源：cs10116 final exam

思路：先拼接出最小值：即字典序最小；要保证每一个小的字符串，左移到合适位置，需要两两比较（刚好是冒泡排序）。这个题目是个不容易的，字符串处理题目。

求minimum时，对相邻两strA[k]与A[k+1]，比较A[k]+A[k+1]与A[k+1]+A[k]的大小，若A[k+1]+A[k]大，颠倒A[k]与A[k+1]；最多交换len(A)-1次。求maximum时，颠倒求minimum时的有序序列即可。使用冒泡排序，循环(n-1)次。

把这些数当成字符串处理，然后采用类似冒泡排序的做法排出大小。

```
# O(n^2)
n = int(input())
nums = input().split()
for i in range(n - 1):
    for j in range(i+1, n):
        #print(i,j)
        if nums[i] + nums[j] < nums[j] + nums[i]:
            nums[i], nums[j] = nums[j], nums[i]

ans = "".join(nums)
nums.reverse()
print(ans + " " + "".join(nums))
```

2020fall-cs101，黄旭

思路：这道题的关键应该是找到排序的方式，前一个数和后一个数比较，如果位数不足，就要重新从第一位开始比，所以说我就先取这个数列的最大位数，然后把每个数都扩充到相同位数进行比较，就可以了。

```
# 虽然能AC，但实际上不对。两倍长度是正确的。
from math import ceil
input()
lt = input().split()

max_len = len(max(lt, key = lambda x:len(x)))
lt.sort(key = lambda x: tuple([int(i) for i in x]) * ceil(max_len/len(x)))
lt1 = lt[::-1]
print(''.join(lt1),''.join(lt))
```

```
# 两倍长度是正确的。O(nlogn)
from math import ceil
input()
lt = input().split()

max_len = len(max(lt, key = lambda x:len(x)))
lt.sort(key = lambda x: x * ceil(2*max_len/len(x)))
lt1 = lt[::-1]
print(''.join(lt1),''.join(lt))
```

CF1364A. XXXXX

https://codeforces.com/problemset/problem/1364/A

brute force/data structures/number theory/two pointers, 1200, https://codeforces.com/problemset/problem/1364/A

Ehab loves number theory, but for some reason he hates the number x. Given an array a, find the length of its longest subarray such that the sum of its elements **isn't** divisible by x, or determine that such subarray doesn't exist.

An array a is a subarray of an array b if a can be obtained from b by deletion of several (possibly, zero or all) elements from the beginning and several (possibly, zero or all) elements from the end.

Input

The first line contains an integer t ($1 \le t \le 5$) — the number of test cases you need to solve. The description of the test cases follows.

The first line of each test case contains 2 integers n and x ($1 \le n \le 10^5$, $1 \le x \le 10^4$) — the number of elements in the array a and the number that Ehab hates.

The second line contains n space-separated integers $a_1, a_2, \ldots\ldots, a_n (0 \le a_i \le 10^4)$ — the elements of the array a.

Output

For each testcase, print the length of the longest subarray whose sum isn't divisible by x. If there's no such subarray, print −1.

Example

input

```
3
3 3
1 2 3
3 4
1 2 3
2 2
0 6
```

output

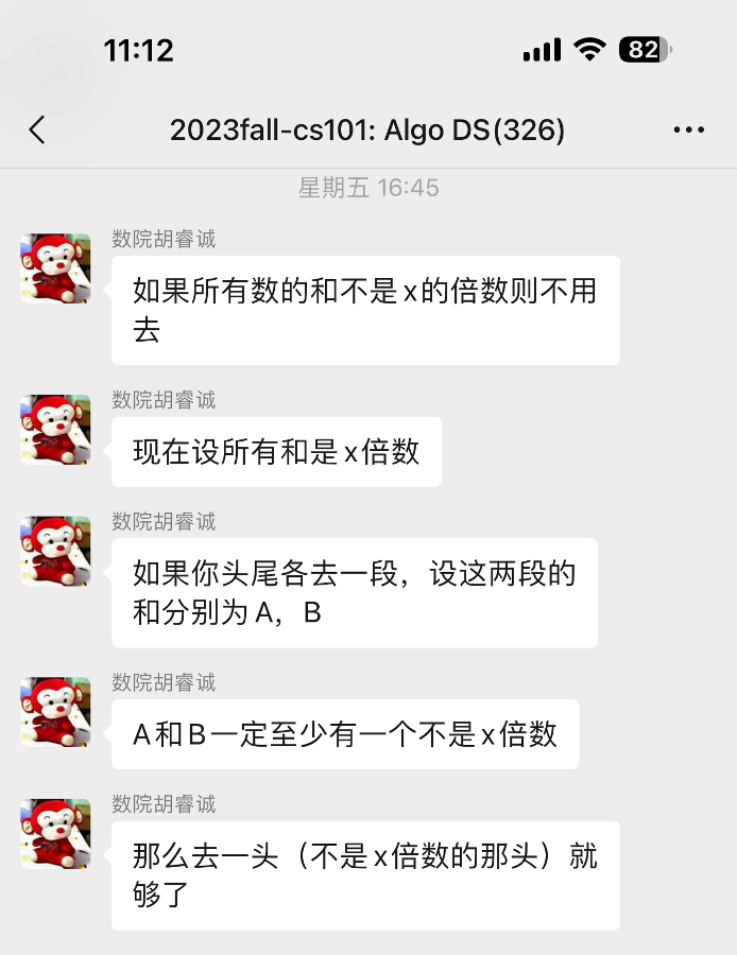
```
2
3
-1
```

Note

In the first test case, the subarray [2,3] has sum of elements 5, which isn't divisible by 3.

In the second test case, the sum of elements of the whole array is 6, which isn't divisible by 4.

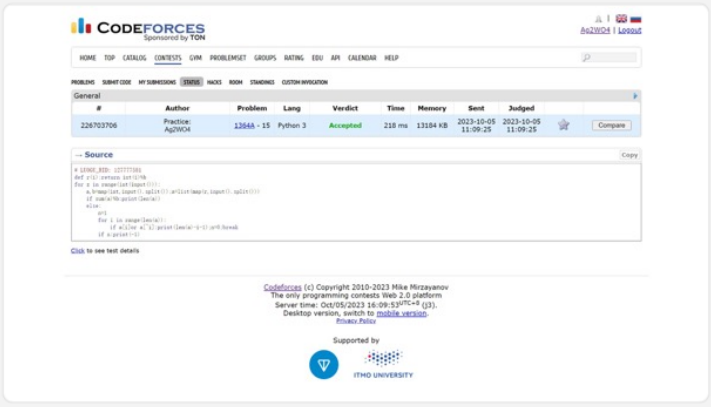
In the third test case, all subarrays have an even sum, so the answer is −1.




```python
# 查达闻
def r(i):return int(i)%b
for z in range(int(input())):
  a,b=map(int,input().split());a=list(map(r,input().split()))
  if sum(a)%b:print(len(a))
  else:
    n=1
    for i in range(len(a)):
      if a[i]or a[~i]:print(len(a)-i-1);n=0;break
    if n:print(-1)
```

2023fall-cs101: Algo DS(325)

查达闻



Oct 5, 2023 23:25

数院胡睿诚

(但是我感觉很难看懂)

"23-花园-何雨轩" 拍了拍 "数院胡睿诚"

头大，都是1个字母的

查达闻: def r(i):return int(i)%b for z in range(int(input())): a,b=map(int,input().split())...

```python
def prefix_sum(nums):
    prefix = []
    total = 0
    for num in nums:
        total += num
        prefix.append(total)
    return prefix

def suffix_sum(nums):
    suffix = []
    total = 0
    # 首先将列表反转
    reversed_nums = nums[::-1]
    for num in reversed_nums:
        total += num
        suffix.append(total)
    # 将结果反转回来
```

```
    suffix.reverse()
    return suffix


t = int(input())
for _ in range(t):
    N, x = map(int, input().split())
    a = [int(i) for i in input().split()]
    aprefix_sum = prefix_sum(a)
    asuffix_sum = suffix_sum(a)

    left = 0
    right = N - 1
    if right == 0:
        if a[0] % x !=0:
            print(1)
        else:
            print(-1)
        continue

    leftmax = 0
    rightmax = 0
    while left != right:
        total = asuffix_sum[left]
        if total % x != 0:
            leftmax = right - left + 1
            break
        else:
            left += 1

    left = 0
    right = N - 1
    while left != right:
        total = aprefix_sum[right]
        if total % x != 0:
            rightmax = right - left + 1
            break
        else:
            right -= 1

    if leftmax == 0 and rightmax == 0:
        print(-1)
    else:
        print(max(leftmax, rightmax))
```

四、Matrices 矩阵

在学习编程的过程中，经常遇到输入的数据是矩阵的形式，所以我们首先来明确矩阵的概念。

4.1 知识点：矩阵

这段矩阵知识点的讲解，借鉴自《数学要素》的1.4和1.5节，作者：姜伟生，2023-06-01出版。

万物皆数。
All is Number.

——毕达哥拉斯(Pythagoras)｜古希腊哲学家、数学家｜570 B.C.—495 B.C.

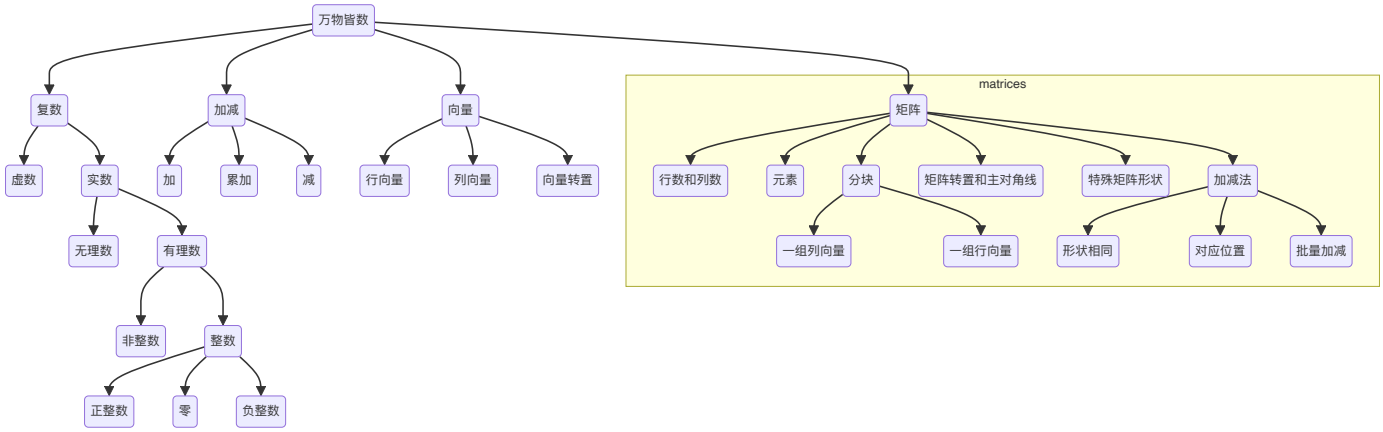


图1 数的结构

4.1.1 向量：数字排成行、列

向量、矩阵等线性代数概念对于数据科学和机器学习至关重要。在机器学习中，数据几乎都以矩阵形式存储、运算。毫不夸张地说，没有线性代数就没有现代计算机运算。逐渐地，大家会发现算数、代数、解析几何、微积分、概率统计、优化方法并不是一个个孤岛，而线性代数正是连接它们的重要桥梁之一。

行向量、列向量

若干数字排成一行或一列，并且用中括号括起来，得到的数组叫作向量(vector)。
排成一行的叫作行向量(row vector)，排成一列的叫作列向量(column vector)。
通俗地讲，行向量就是表格的一行数字，列向量就是表格的一列数字。以下两例分别展示了行向量和列向量，即

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}_{1\times 3}, \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}_{3\times 1} \tag{1}$$

式(1)中，下角标“1×3”代表“1行、3列”，“3×1”代表“3行、1列”。

转置

转置符号为上标“T”。行向量转置(transpose)可得到列向量；同理，列向量转置可得到行向量。举例如下，有

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}^{\mathrm{T}} = \begin{bmatrix}1\\2\\3\end{bmatrix}, \begin{bmatrix}1\\2\\3\end{bmatrix}^{\mathrm{T}} = \begin{bmatrix}1 & 2 & 3\end{bmatrix} \tag{2}$$

4.1.2 矩阵：数字排列成长方形

矩阵(matrix)将一系列数字以长方形方式排列，如

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}_{2\times 3}, \begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix}_{3\times 2}, \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}_{2\times 2} \tag{3}$$

通俗地讲，矩阵将数字排列成表格，有行、有列。式(3)给出了三个矩阵，形状分别是2行3列（记作2×3）、3行2列（记作3×2）和2行2列（记作2×2）。

通常用大写字母代表矩阵，比如矩阵A和矩阵B。

图2所示为一个n×D矩阵X。n是矩阵的行数(number of rows in the matrix)，D是矩阵的列数(number of columns in the matrix)。X可以展开写成表格形式，即

$$X_{n\times D} = \begin{bmatrix}x_{1,1} & x_{1,2} & \cdots & x_{1,D}\\x_{2,1} & x_{2,2} & \cdots & x_{2,D}\\\vdots & \vdots & \ddots & \vdots\\x_{n,1} & x_{n,2} & \cdots & x_{n,D}\end{bmatrix} \tag{4}$$

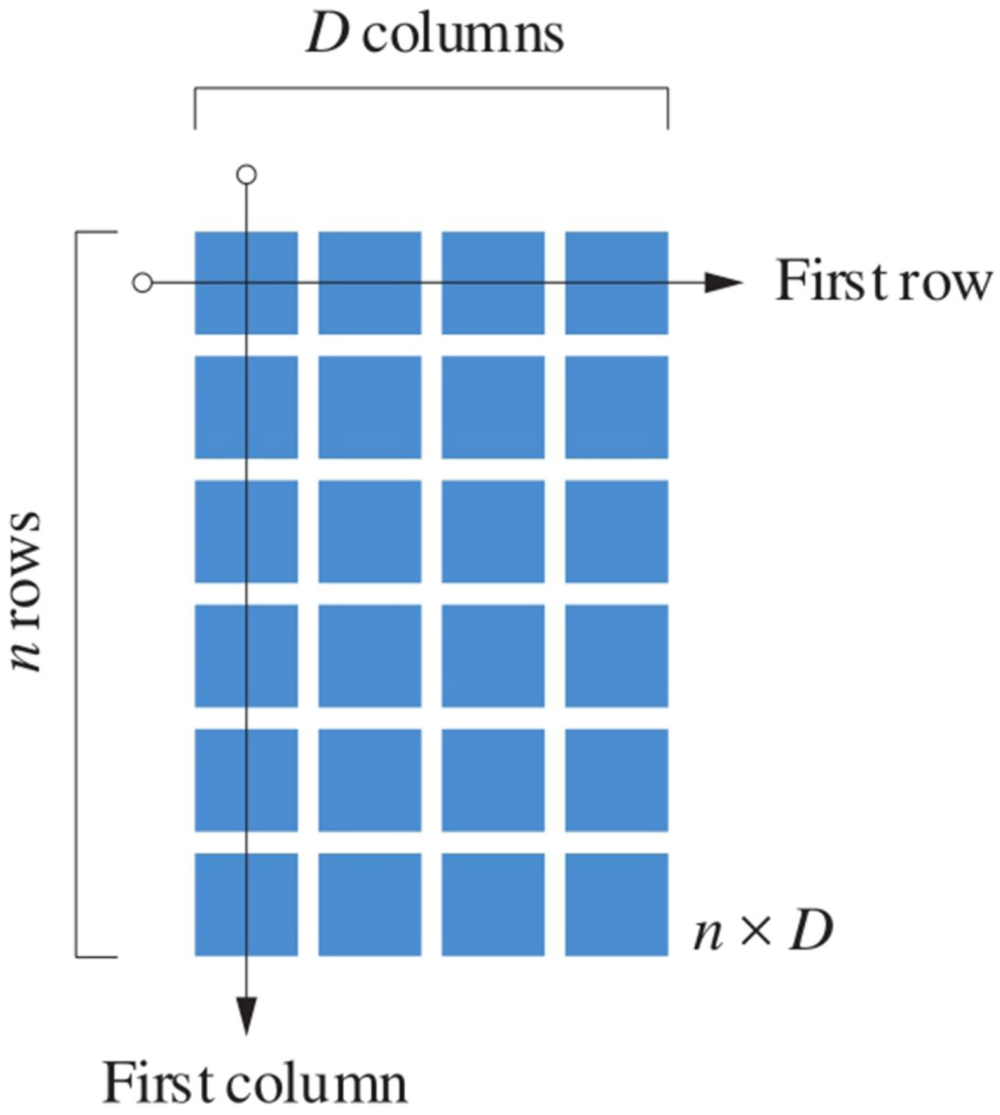


图2　n×D矩阵X

再次强调：先说行序号，再说列序号。数据矩阵一般采用大写X表达。

矩阵X中，元素(element) $x_{i,j}$ 被称作i,j元素（i j entry或i j element），也可以说$x_{i,j}$出现在i行j列(appears in row i and column j)。比如，$x_{n,1}$是矩阵X的第n行、第1列元素。
表1.4总结了如何用英文读矩阵和矩阵元素。

表1.4　矩阵有关英文表达

数学表达	英文表达
$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	Two by two matrix, first row one two, second row three four
$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$	*m* by *n* matrix, first row *a* sub one one, *a* sub one two, dot dot dot, *a* sub one n, second row *a* sub two one, *a* sub two two, dot dot dot, *a* sub two *n* dot dot dot last row *a* sub *m* one, *a* sub *m* two, dot dot dot *a* sub *m n*
$a_{i,j}$	Lowercase (small) *a* sub *i* comma *j*
$a_{i,j+1}$	Lowercase *a* double subscript *i* comma *j* plus one
$a_{i,j-1}$	Lowercase *a* double subscript *i* comma *j* minus one

4.2 编程题目

4.2.1 保护圈

12560: 生存游戏

matrices, http://cs101.openjudge.cn/practice/12560/

有如下生存游戏的规则：

给定一个n*m(1<=n,m<=100)的数组，每个元素代表一个细胞，其初始状态为活着(1)或死去(0)。

每个细胞会与其相邻的8个邻居（除数组边缘的细胞）进行交互，并遵守如下规则：

任何一个活着的细胞如果只有小于2个活着的邻居，那它就会由于人口稀少死去。

任何一个活着的细胞如果有2个或者3个活着的邻居，就可以继续活下去。

任何一个活着的细胞如果有超过3个活着的邻居，那它就会由于人口拥挤而死去。

任何一个死去的细胞如果有恰好3个活着的邻居，那它就会由于繁殖而重新变成活着的状态。

请写一个函数用来计算所给定初始状态的细胞经过一次更新后的状态是什么。

注意：所有细胞的状态必须同时更新，不能使用更新后的状态作为其他细胞的邻居状态来进行计算。

输入

第一行为n和m，而后n行，每行m个元素，用空格隔开。

输出

n行，每行m个元素，用空格隔开。

样例输入

```
3 4
0 0 1 1
1 1 0 0
1 1 0 1
```

样例输出

```
0 1 1 0
1 0 0 1
1 1 1 0
```

来源：cs10116 final exam

```python
# http://cs101.openjudge.cn/practice/12560/
def check(board, y, x):
    c = board[y-1][x-1]+board[y-1][x]+ board[y-1][x+1]+ \
        board[y][x-1]+board[y][x+1]+ \
        board[y+1][x-1]+board[y+1][x]+board[y+1][x+1]
    if board[y][x] and (c<2 or c>3):
        return 0
    elif board[y][x]==0 and c==3:
        return 1

    return board[y][x]

n,m=[int(i) for i in input().split()]

board=[]
board.append( [0 for x in range(m+2)] )
for y in range(n):
    board.append([0] +[int(x) for x in input().split()] + [0])

board.append( [0 for x in range(m+2)] )


bn = [[0]*m for y in range(n)]
for y in range(n):
    for x in range(m):
        bn[y][x] = check(board, y+1, x+1)

for y in range(n):
    print(' '.join([str(x) for x in bn[y] ]))
```

函数

保护圈

' '.join(map(str, bn[y])

加保护圈，八个邻居步长用dx,dy对表示。

```python
dx = [-1, -1, -1, 0, 1, 1,  1,  0]
```

```
dy = [-1,  0,  1, 1, 1, 0, -1, -1]

def check(board, y, x):
    c = 0
    for i in range(8):
        nx = x + dx[i]
        ny = y + dy[i]
        c += board[ny][nx]

    if board[y][x] and (c<2 or c>3):
        return 0
    elif board[y][x]==0 and c==3:
        return 1

    return board[y][x]

n, m = map(int, input().split())

board=[]
board.append( [0 for x in range(m+2)] )
for _ in range(n):
    board.append([0] +[int(_) for _ in input().split()] + [0])

board.append( [0 for _ in range(m+2)] )

# in place solver
bn = [[0]*m for y in range(n)]
for i in range(n):
    for j in range(m):
        bn[i][j] = check(board, i+1, j+1)

for row in bn:
    print(*row)
```

508A. Pasha and Pixels

brute force, 1100, http://codeforces.com/problemset/problem/508/A

Pasha loves his phone and also putting his hair up... But the hair is now irrelevant.

Pasha has installed a new game to his phone. The goal of the game is following. There is a rectangular field consisting of *n* row with *m* pixels in each row. Initially, all the pixels are colored white. In one move, Pasha can choose any pixel and color it black. In particular, he can choose the pixel that is already black, then after the boy's move the pixel does not change, that is, it remains black. Pasha loses the game when a 2 × 2 square consisting of black pixels is formed.

Pasha has made a plan of *k* moves, according to which he will paint pixels. Each turn in his plan is represented as a pair of numbers *i* and *j*, denoting respectively the row and the column of the pixel to be colored on the current move.

Determine whether Pasha loses if he acts in accordance with his plan, and if he does, on what move the 2 × 2 square consisting of black pixels is formed.

Input

The first line of the input contains three integers n, m, k ($1 \le n, m \le 1000$, $1 \le k \le 10^5$) — the number of rows, the number of columns and the number of moves that Pasha is going to perform.

The next k lines contain Pasha's moves in the order he makes them. Each line contains two integers i and j ($1 \le i \le n$, $1 \le j \le m$), representing the row number and column number of the pixel that was painted during a move.

Output

If Pasha loses, print the number of the move when the 2 × 2 square consisting of black pixels is formed.

If Pasha doesn't lose, that is, no 2 × 2 square consisting of black pixels is formed during the given k moves, print 0.

Examples

input

```
2 2 4
1 1
1 2
2 1
2 2
```

output

```
4
```

input

```
2 3 6
2 3
2 2
1 3
2 2
1 2
1 1
```

output

```
5
```

input

```
5 3 7
2 3
1 2
1 1
4 1
3 1
5 3
3 2
```

output

```
0
```

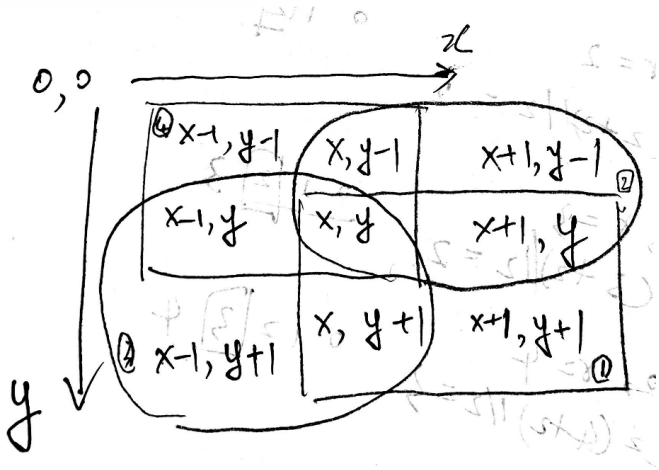
44603553	Practice: GMyhf	508A - 32	Python 3	Accepted

→ Source

```
n,m,k = map(int, input().split())
mx = [(m+2)*[0] for i in range(n+2)]

# if square 2 × 2 formed from black cells appears, and
# cell (i, j) will upper-left, upper-right, bottom-left
# or bottom-right of this squares.

def square_check(i,j):
    if mx[i][j+1] and mx[i+1][j] and mx[i+1][j+1]:
        return True
    if mx[i][j-1] and mx[i+1][j-1] and mx[i+1][j]:
        return True
    if mx[i-1][j] and mx[i-1][j+1] and mx[i][j+1]:
        return True
    if mx[i-1][j-1] and mx[i-1][j] and mx[i][j-1]:
        return True
    return False

for i in range(k):
    x,y = map(int, input().split())
    mx[x][y] = 1
    if square_check(x,y):
        print(i+1)
        break
else:
    print(0)
```

保护圈

函数



练习加保护圈

```
# http://codeforces.com/contest/508/submission/44603553
n,m,k = map(int, input().split())
mx = [(m+2)*[0] for i in range(n+2)]

# if square 2 × 2 formed from black cells appears, and
# cell (i, j) will upper-left, upper-right, bottom-left
# or bottom-right of this squares.

```

```python
def square_check(i,j):
    if mx[i][j+1] and mx[i+1][j] and mx[i+1][j+1]:
        return True
    if mx[i][j-1] and mx[i+1][j-1] and mx[i+1][j]:
        return True
    if mx[i-1][j] and mx[i-1][j+1] and mx[i][j+1]:
        return True
    if mx[i-1][j-1] and mx[i-1][j] and mx[i][j-1]:
        return True
    return False

for i in range(k):
    x,y = map(int, input().split())
    mx[x][y] = 1
    if square_check(x,y):
        print(i+1)
        break
else:
    print(0)
```

4.2.2 range中使用min、max

02659:Bomb Game

matrices, http://cs101.openjudge.cn/practice/02659/

Bosko and Susko are playing an interesting game on a board made of rectangular fields arranged in A rows and B columns.

When the game starts, Susko puts its virtual pillbox in one field one the board. Then Bosko selects fields on which he will throw his virtual bombs. After each bomb, Susko will tell Bosko whether his pillbox is in the range of this bomb or not.

The range of a bomb with diameter P (P is always odd), which is thrown in field (R, S), is a square area. The center of the square is in the field (R, S), and the side of the square is parallel to the sides of the board and with length P.

After some bombs have been thrown, Bosko should find out the position of Susko's pillbox. However, the position may be not unique, and your job is to help Bosko to calculate the number of possible positions.

输入

First line of input contains three integers: A, B and K, 1 <= A, B, K <=100. A represents the number of rows, B the number of columns and K the number of thrown bombs.

Each of the next K lines contains integers R, S, P and T, describing a bomb thrown in the field at R-th row and S-th column with diameter P, 1 <= R <= A, 1 <= S <= B, 1 <= P <= 99, P is odd. If the pillbox is in the range of this bomb, T equals to 1; otherwise it is 0.

输出

Output the number of possible fields, which Susko's pillbox may stay in.

样例输入

```
5 5 3
3 3 3 1
3 4 1 0
3 4 3 1
```

样例输出

```
5
```

来源

Croatia OI 2002 National – Juniors

```python
def max_count(matrix):
    maximum = max(max(row) for row in matrix)
    count = sum(row.count(maximum) for row in matrix)
    return count

def calculate_possible_positions(A, B, K, bombs):
    positions = [[0] * B for _ in range(A)]

    for (R, S, P, T) in bombs:
        for i in range(max(0, R - (P - 1) // 2), min(A, R + (P + 1) // 2)):
            for j in range(max(0, S - (P - 1) // 2), min(B, S + (P + 1) // 2)):
                if T == 1 :
                    positions[i][j] += 1

                elif T == 0:
                    positions[i][j] -= 1

    #for row in positions:
    #    print(row)
    return max_count(positions)

A, B, K = map(int, input().split())
bombs = []
for _ in range(K):
    R, S, P, T = map(int, input().split())
```

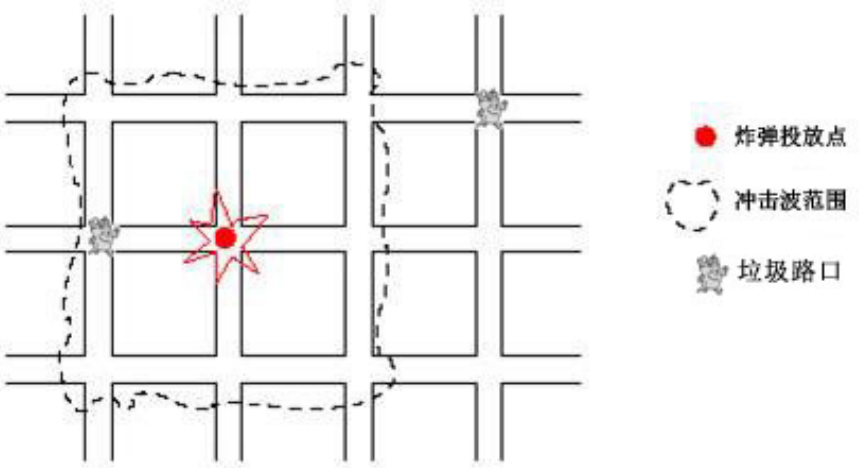
```
    bombs.append((R - 1, S - 1, P, T))

result = calculate_possible_positions(A, B, K, bombs)
print(result)
```

04133:垃圾炸弹

matrices, http://cs101.openjudge.cn/practice/04133/

2018年俄罗斯世界杯（2018 FIFA World Cup）开踢啦！为了方便球迷观看比赛，莫斯科街道上很多路口都放置了的直播大屏幕，但是人群散去后总会在这些路口留下一堆垃圾。为此俄罗斯政府决定动用一种最新发明——“垃圾炸弹”。这种“炸弹”利用最先进的量子物理技术，爆炸后产生的冲击波可以完全清除波及范围内的所有垃圾，并且不会产生任何其他不良影响。炸弹爆炸后冲击波是以正方形方式扩散的，炸弹威力（扩散距离）以d给出，表示可以传播d条街道。

例如下图是一个d=1的“垃圾炸弹”爆炸后的波及范围。



图：“垃圾炸弹”爆炸冲击波范围

假设莫斯科的布局为严格的1025*1025的网格状，由于财政问题市政府只买得起一枚“垃圾炸弹”，希望你帮他们找到合适的投放地点，使得一次清除的垃圾总量最多（假设垃圾数量可以用一个非负整数表示，并且除设置大屏幕的路口以外的地点没有垃圾）。

输入

第一行给出“炸弹”威力d(1 <= d <= 50)。第二行给出一个数组n(1 <= n <= 20)表示设置了大屏幕(有垃圾)的路口数目。接下来n行每行给出三个数字x, y, i, 分别代表路口的坐标(x, y)以及垃圾数量i. 点坐标(x, y)保证是有效的（区间在0到1024之间），同一坐标只会给出一次。

输出

输出能清理垃圾最多的投放点数目，以及能够清除的垃圾总量。

样例输入

```
1
2
4 4 10
6 6 20
```

样例输出

```
1 30
```

```python
#gpt
'''
过遍历方式计算出在每个点投掷炸弹能清理的垃圾数量，并用max_point存储垃圾数量的最大值，
res存储清理垃圾数量最大时的点的数量。最后输出结果。
是一个比较经典的滑动窗口问题
'''
d = int(input())
n = int(input())
square = [[0]*1025 for _ in range(1025)]
for _ in range(n):
    x, y, k = map(int, input().split())
    #for i in range(x-d if x-d >= 0 else 0, x+d+1 if x+d <= 1024 else 1025):
      #for j in range(y-d if y-d >= 0 else 0, y+d+1 if y+d <= 1024 else 1025):
    for i in range(max(x-d, 0), min(x+d+1, 1025)):
        for j in range(max(y-d, 0), min(y+d+1, 1025)):
          square[i][j] += k

res = max_point = 0
for i in range(0, 1025):
  for j in range(0, 1025):
    if square[i][j] > max_point:
      max_point = square[i][j]
      res = 1
    elif square[i][j] == max_point:
      res += 1
print(res, max_point)
```

五、其他知识点

5.1 Regular expression 甜点

不要求必须掌握，但是会了可以用，有时候很便捷。

正则表达式是对字符串操作的一种逻辑公式，就是用事先定义好的一些特定字符、及这些特定字符的组合，组成一个“规则字符串”，这个“规则字符串”用来表达对字符串的一种过滤逻辑。

正则表达式，又称规则表达式，（Regular Expression，在代码中常简写为regex、regexp或RE），是一种文本模式，包括普通字符（例如，a 到 z 之间的字母）和特殊字符（称为"元字符"），是计算机科学的一个概念。正则表达式使用单个字符串来描述、匹配一系列匹配某个句法规则的字符串，通常被用来检索、替换那些符合某个模式（规则）的文本。

Regulex正则表达式在线测试工具，https://regex101.com

Python正则表达式详解

https://blog.csdn.net/weixin_43347550/article/details/105158003

04015: 邮箱验证

strings, http://cs101.openjudge.cn/practice/04015

POJ 注册的时候需要用户输入邮箱，验证邮箱的规则包括：
1)有且仅有一个'@'符号
2)'@'和'.'不能出现在字符串的首和尾
3)'@'之后至少要有一个'.'，并且'@'不能和'.'直接相连
满足以上3条的字符串为合法邮箱，否则不合法，
编写程序验证输入是否合法

输入

输入包含若干行，每一行为一个代验证的邮箱地址，长度小于100

输出

每一行输入对应一行输出
如果验证合法，输出 YES
如果验证非法：输出 NO

样例输入

```
.a@b.com
pku@edu.cn
cs101@gmail.com
cs101@gmail
```

样例输出

```
NO
YES
YES
NO
```

这题目输入没有明确结束，需要套在try ... except里面。测试时候，需要模拟输入结束，看你是window还是mac。If the user hits EOF (*nix: Ctrl-D, Windows: Ctrl-Z+Return), raise EOFError.

题目给的要求是[^@.]，也就是说正常字段只需要不是“@”和“.”即可。以前遇到的要求是：正常字段只能是大小写字母或“-”，所以也试了试[\w-]。虽然regulation需要前后match，也就是说前面加一个“^”，后面加一个“$”， 但是.match函数本身就是从头开始检索的，所以“^”可以删去。

```python
# https://www.tutorialspoint.com/python/python_reg_expressions.htm
# https://www.geeksforgeeks.org/python-regex/

import re
while True:
    try:
        s = input()
        reg = r'[\w-]+(\.[\w-]+)*@[\w-]+(\.[\w-]+)+$'
        print('YES' if re.match(reg, s) else 'NO')
    except EOFError:
        break
```

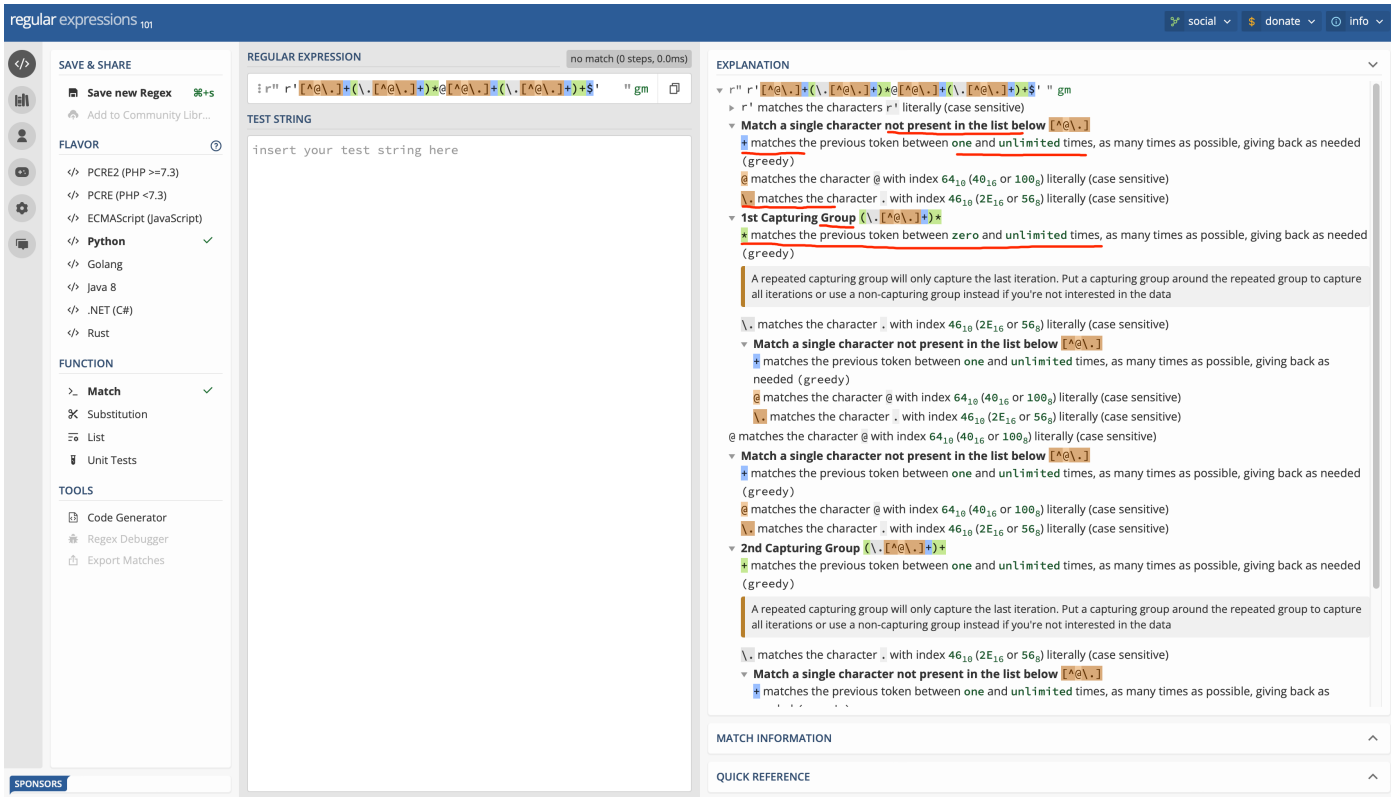
```python
# https://www.tutorialspoint.com/python/python_reg_expressions.htm
# https://www.geeksforgeeks.org/python-regex/
import re
while True:
    try:
        s = input()
        reg   = r'[^@\.]+(\.[^@\.]+)*@[^@\.]+(\.[^@\.]+)+$'
        print('YES' if re.match(reg, s) else 'NO')
    except EOFError:
        break
```

[^xyz]，匹配未包含的任意字符。例如，“[^abc]”可以匹配“plain”中的“plin”任一字符。

$匹配输入行尾。

(pattern)，匹配pattern并获取这一匹配。所获取的匹配可以从产生的Matches集合得到。

https://regex101.com



24834: 通配符匹配

http://cs101.openjudge.cn/practice/24834/

给定一个字符串s和一个字符模式p，请实现一个支持'?'和'*'的通配符匹配功能。

其中‘?’可以匹配任何单个字符，如‘a?c’可以成功匹配‘aac’,‘abc’等字符串，但不可匹配‘ac’,‘aaac’等字符串 。

‘*’ 可以匹配任意长度字符串（包括空字符串）,如‘a*c’可以成功匹配‘ac’,‘abdc’,‘abc’,‘aaac’等字符串，但不可匹配‘acb’，‘cac’等字符串。

两个字符串完全匹配才算匹配成功。

输入

输入为一个数字n表示测试字符串与字符模式对数，换行。(n ≤ 30)
后续2n行为每组匹配的s与p，每行字符串后换行。
s 非空，只包含从 a-z 的小写字母。
p 非空，只包含从 a-z 的小写字母，以及字符？和 *。
字符串s和p的长度均小于50

输出

每一组匹配串匹配成功输出‘yes’,否则输出‘no’。

样例输入

```
3
abc
abc
abc
a*c
abc
a??c
```

样例输出

```
yes
yes
no
```

```python
#23n2300017735(夏天明BrightSummer)
import re

for i in range(int(input())):
    s, p = input(), input().replace("?", ".{1}").replace("*", ".*") + "$"
    print("yes" if re.match(p, s) else "no")
```

.点，匹配除“\n”和"\r"之外的任何单个字符。要匹配包括“\n”和"\r"在内的任何字符，请使用像“[\s\S]”的模式。

*，匹配前面的子表达式任意次。例如，z*能匹配“z”，也能匹配“zo”以及“zoo”*。等价于{0,}。

58A. Chat room

greedy/strings, 1000, http://codeforces.com/problemset/problem/58/A

Vasya has recently learned to type and log on to the Internet. He immediately entered a chat room and decided to say hello to everybody. Vasya typed the word *s*. It is considered that Vasya managed to say hello if several letters can be deleted from the typed word so that it resulted in the word "hello". For example, if Vasya types the word "ahhellllloou", it will be considered that he said hello, and if he types "hlelo", it will be considered that Vasya got misunderstood and he didn't manage to say hello. Determine whether Vasya managed to say hello by the given word *s*.

Input

The first and only line contains the word *s*, which Vasya typed. This word consisits of small Latin letters, its length is no less that 1 and no more than 100 letters.

Output

If Vasya managed to say hello, print "YES", otherwise print "NO".

Examples

input

```
ahhellllloou
```

output

```
YES
```

input

```
hlelo
```

output

```
NO
```

```
import re
s = input()
r = re.search('h.*e.*l.*l.*o', s)
print(['YES', 'NO'][r==None])
```

LeetCode 65. 有效数字

https://leetcode.cn/problems/valid-number/description/

https://leetcode.cn/problems/valid-number/solutions/564188/you-xiao-shu-zi-by-leetcode-solution-298l/

这个正则表达式 pattern 用于判断一个字符串是否是有效数字。下面我来详细解释一下其中的各个部分：

- `^` 表示匹配字符串的开始位置。
- `[-+]?` 表示一个可选的符号字符，可以是正号 `+` 或负号 `-`。
- `(\d+(\.\d*)?|\.\d+)` 表示有效数字的主要部分，可以分成三种情况：
  - `\d+(\.\d*)?` 表示至少一位数字，后面可选的小数部分，小数部分可以没有或有多个小数位。
  - `|` 表示或的关系。
  - `.\d+` 表示以点 `.` 开始，后面至少一位数字的小数形式。
- `([eE][-+]?\d+)?` 表示指数部分，也是一个可选项，可以是 `e` 或 `E` 开头，后面可选的符号字符，以及至少一位数字。
- `$` 表示匹配字符串的结束位置。

综合起来，整个正则表达式可以解释为：

- 首先可以匹配一个可选的符号字符。
- 接下来是有效数字的主要部分，可以是整数或小数形式。
- 最后是可选的指数部分。

因此，该正则表达式可以匹配符合有效数字要求的字符串。在 Python 中使用 `re.match` 方法进行匹配时，如果匹配成功，说明字符串是一个有效数字，返回 `True`；否则，返回 `None`，表示不是一个有效数字。

```python
class Solution:
    def isNumber(self, s: str) -> bool:
        import re
        pattern = r'^[-+]?(\d+(\.\d*)?|\.\d+)([eE][-+]?\d+)?$'

        ans = re.match(pattern, s)
        if ans is not None:
            return True
        else:
            return False
```

https://stackoverflow.com/questions/43233535/explicitly-define-datatype-in-python-function

22

Python is a strongly-typed dynamic language, which associates types with *values*, not names. If you want to force callers to provide data of specific types the only way you can do so is by adding explicit checks inside your function.

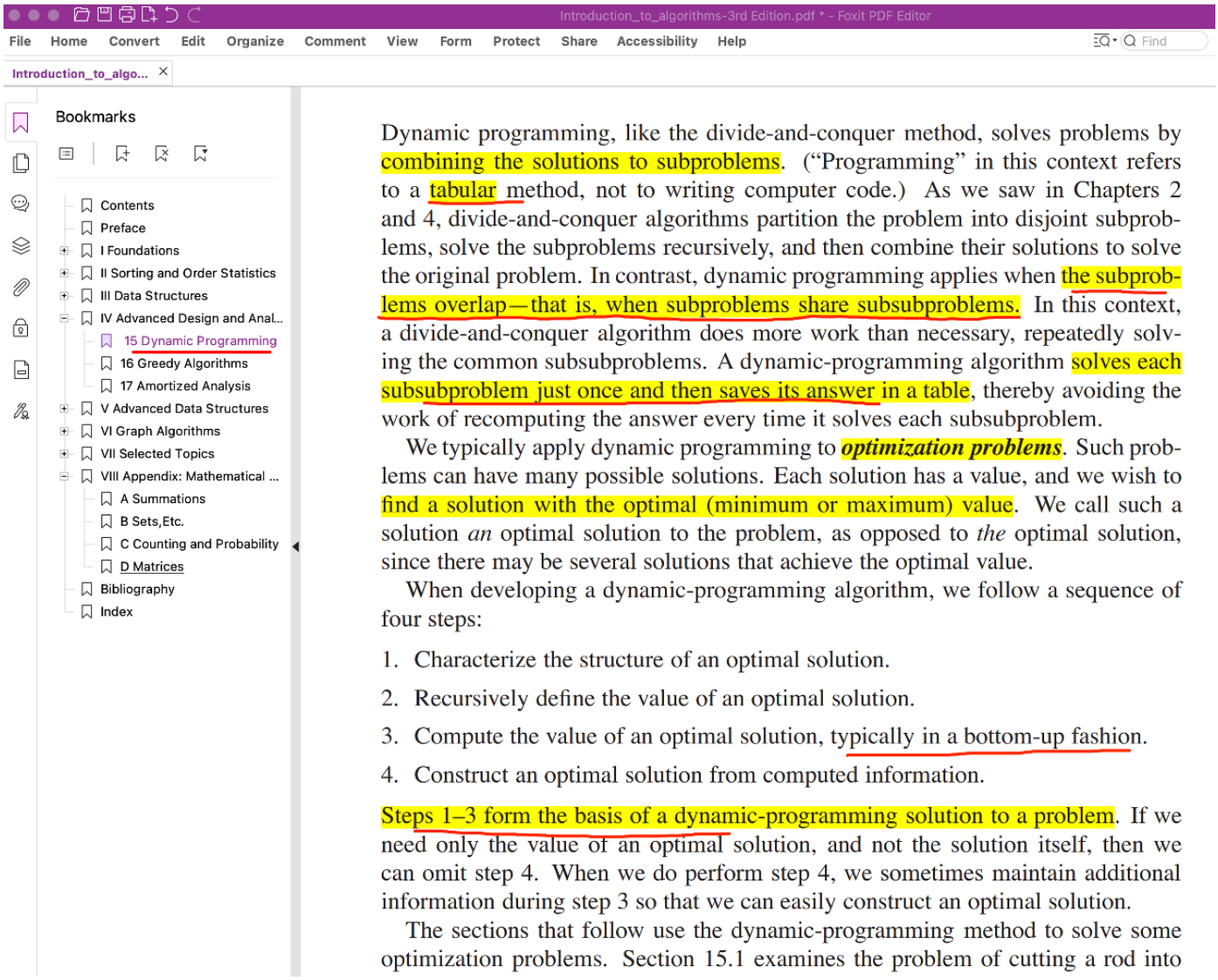
Fairly recently *type annotations* were added to the language. and now you can write syntactically correct function specifications including the types of arguments and return values. The annotated version for your example would be

```
def add(x: float, y: float) -> float:
    return x+y
```

5.2 DP的影子

优化问题除了使用时间复杂度更低的算法（如：线性筛/欧拉筛），还可以用DP。

from functools import lru_cache; lru_cache(maxsize = None)



Dynamic programming, like the divide-and-conquer method, solves problems by combining the solutions to subproblems. ("Programming" in this context refers to a tabular method, not to writing computer code.) As we saw in Chapters 2 and 4, divide-and-conquer algorithms partition the problem into disjoint subproblems, solve the subproblems recursively, and then combine their solutions to solve the original problem. In contrast, dynamic programming applies when the subproblems overlap—that is, when subproblems share subsubproblems. In this context, a divide-and-conquer algorithm does more work than necessary, repeatedly solving the common subsubproblems. A dynamic-programming algorithm solves each subsubproblem just once and then saves its answer in a table, thereby avoiding the work of recomputing the answer every time it solves each subsubproblem.

We typically apply dynamic programming to ***optimization problems***. Such problems can have many possible solutions. Each solution has a value, and we wish to find a solution with the optimal (minimum or maximum) value. We call such a solution *an* optimal solution to the problem, as opposed to *the* optimal solution, since there may be several solutions that achieve the optimal value.

When developing a dynamic-programming algorithm, we follow a sequence of four steps:

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution, typically in a bottom-up fashion.
4. Construct an optimal solution from computed information.

Steps 1–3 form the basis of a dynamic-programming solution to a problem. If we need only the value of an optimal solution, and not the solution itself, then we can omit step 4. When we do perform step 4, we sometimes maintain additional information during step 3 so that we can easily construct an optimal solution.

The sections that follow use the dynamic-programming method to solve some optimization problems. Section 15.1 examines the problem of cutting a rod into

"Programming" 指的是一种表格法，不是写计算机程序

•Dp应用于自问题重叠的情况，即不同的子问题具有公共的子问题

•Dp对每个子问题只求解一次，并将其解保存在一个表格中，从而无需每次求解一个子问题时都重新计算

•最优化问题（optimization problems）可以有很多可行解，每个解都有一个值，希望寻找具有最优值（最小值或最大值）的解。

•称这样的解为问题的一个最优解（an optimal solution），而不是最优解（the optimal solution），因为可能有多个解都达到最优值

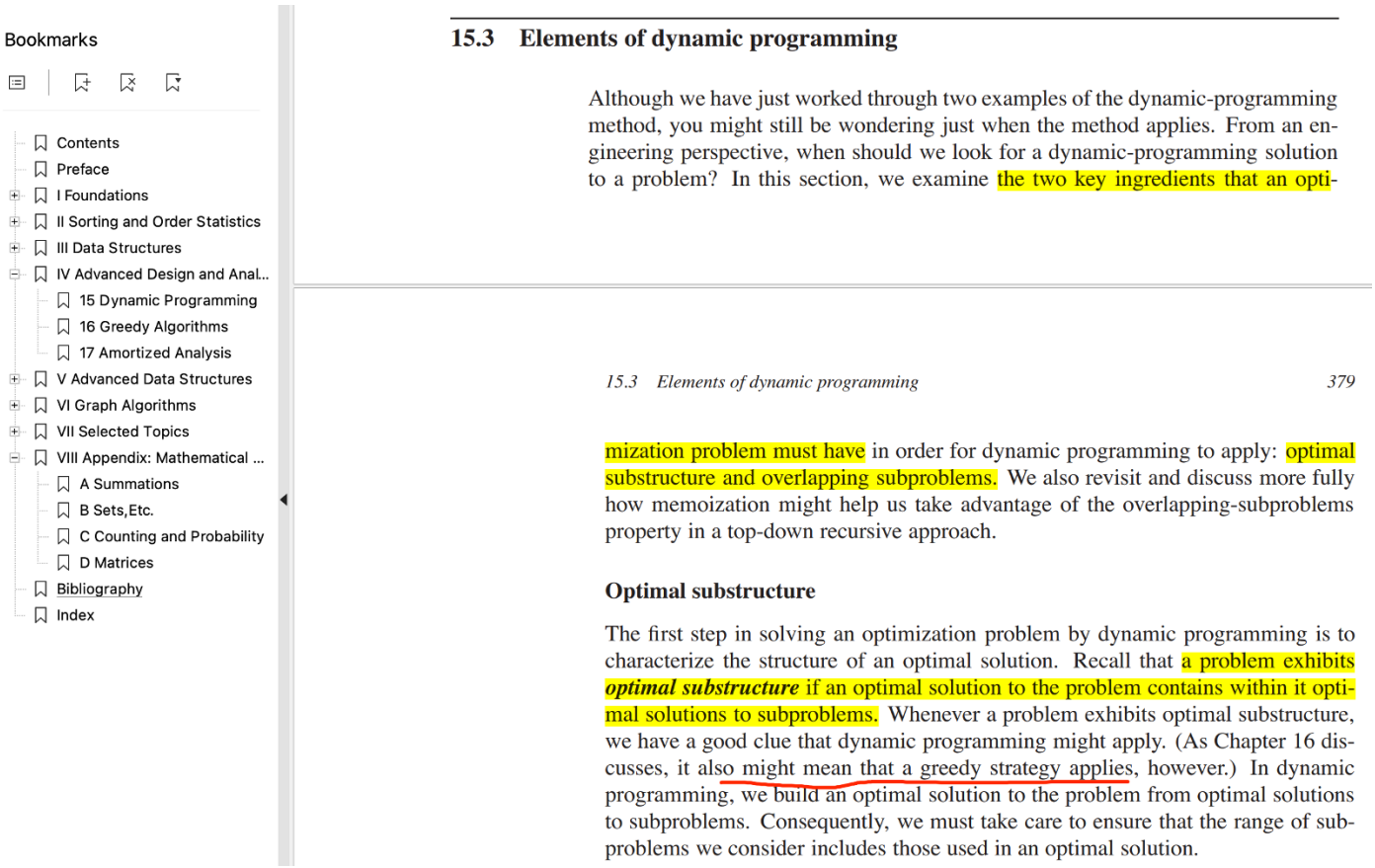
•四个步骤来设计一个dp算法：

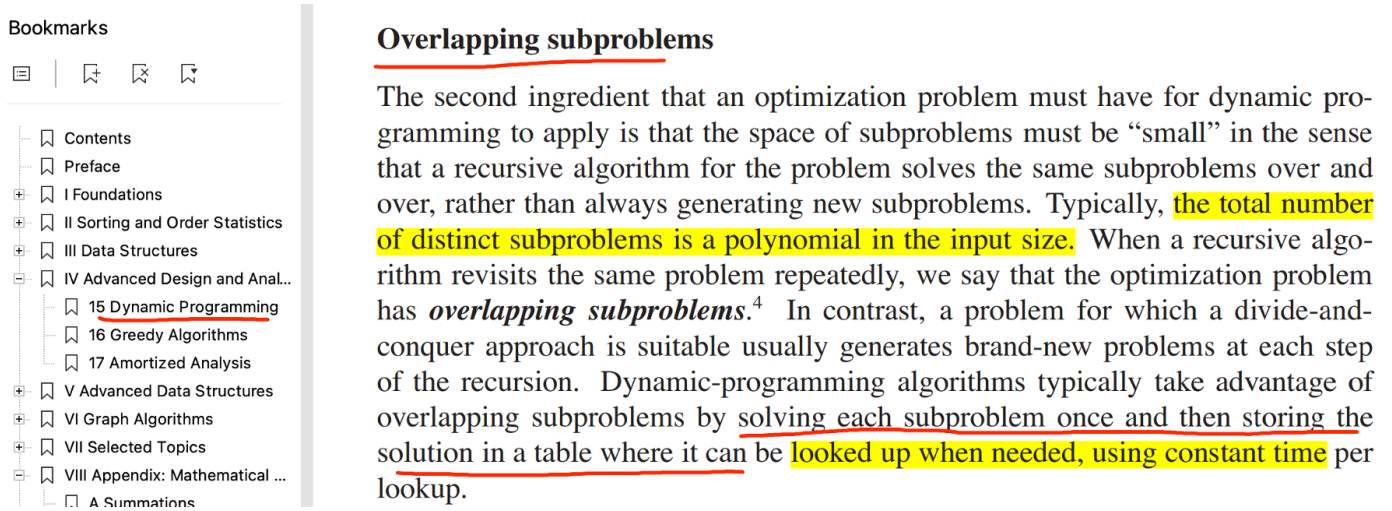
•刻画一个最优解的结构特征

•递推地定义最优解的值

•计算最优的值，通常采用自底向上的方法

•利用计算出的信息结构构造一个最优解





230B. T-primes

binary search, implementation, math, number theory, 1300, http://codeforces.com/problemset/problem/230/B

求解素数的三种方法，包括：试除法（trial division）、埃氏筛（Sieve of Eratosthenes）、欧拉筛（Sieve of Euler，线性法），https://blog.dotcpp.com/a/69737

@lru_cache(maxsize = None)

除余法，pypy3可以AC 230B。lru_cache 如果屏蔽了，超时。

Contest status

#	When	Who	Problem	Lang	Verdict	Time	Memory
227551469	Oct/10/2023 22:21 UTC+8	GMyhf	230B - T-primes	PyPy 3-64	Time limit exceeded on test 63	2000 ms	19100 KB
227551224	Oct/10/2023 22:20 UTC+8	GMyhf	230B - T-primes	PyPy 3-64	Accepted	998 ms	20400 KB

```python
import math
from functools import lru_cache

@lru_cache(maxsize = None)
def prime(x):
    for i in range(2, int(math.sqrt(x))+1):
        if x % i == 0:
            return False
    return True

input()

*a, = map(int, input().split())
ans = []
for i in a:
    if i == 1:
        ans.append('NO')
        continue
    tmp = int(math.sqrt(i))
    if tmp**2 != i:
        ans.append('NO')
        continue

    if prime(tmp):
        ans.append('YES')
    else:
        ans.append('NO')

print('\n'.join(ans))
```

02810: 完美立方

1）lru_cache 有作用，时间接近先算好的方法。完美立方，http://cs101.openjudge.cn/practice/02810/ 2）今天课件里面用lru_cache的程序没有写对，因为它对函数的参数起缓存作用，所以作用的函数一定要有参数。

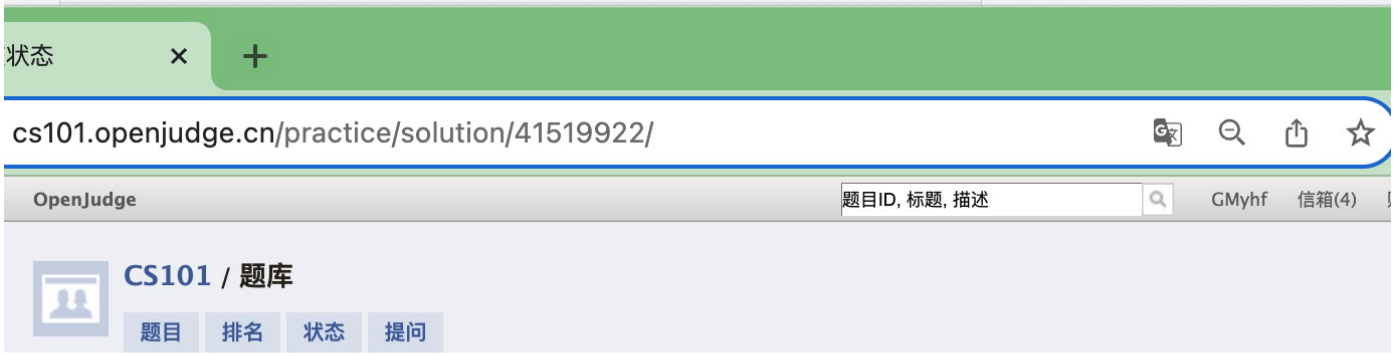
状态: Accepted

源代码

```
from functools import lru_cache

@lru_cache(maxsize = 128)
def cube(i):
    return i**3

def solv():
    N = int(input())
    ans = []
    for a in range(2,N+1):
        for b in range(2,a):
            for c in range(b,a):
                for d in range(c,a):
                    #if a ** 3 == b ** 3 + c ** 3 + d ** 3:
                    if cube(a) == cube(b) + cube(c) + cube(d):
                        #print('Cube = %d, Triple = (%d,%d,%d)' %(a,b,c,
                        ans.append((a,b,c,d))

    return ans

for a,b,c,d in solv():
    print(f"Cube = {a}, Triple = ({b},{c},{d})")
```

基本信息

#: 41573897
题目: 02810
提交人: HFYan(GMyhf)
内存: 3652kB
时间: 1352ms
语言: Python3
提交时间: 2023-10-10 23:29:55



#41519922提交状态

查看 提交 统计

状态: Accepted

源代码

```
n = int(input())
cube = [i**3 for i in range(n+1)]
for a in range(3,n+1):
    for b in range(2,a):
        for c in range(b,a):
            for d in range(c,a):
                if cube[a]==cube[b]+cube[c]+cube[d]:
                    print(f"Cube = {a}, Triple = ({b},{c},{d})")
```

基本信息

#: 41519922
题目: 02810
提交人: HFYan(GMyhf)
内存: 3656kB
时间: 875ms
语言: Python3
提交时间: 2023-10-08 10:33:52

5.3 Data Structure

三个常用的数据结构：stack, queue, heap

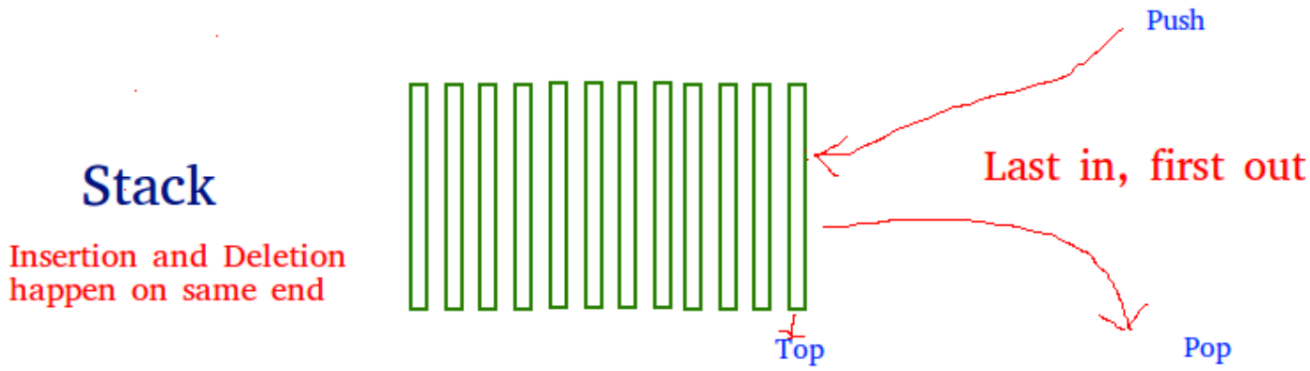
Stack in Python

https://www.geeksforgeeks.org/stack-in-python/

Stack in Python

Read Discuss Courses Practice

A **stack** is a linear data structure that stores items in a **Last-In/First-Out (LIFO)** or First-In/Last-Out (FILO) manner. In stack, a new element is added at one end and an element is removed from that end only. The insert and delete operations are often called push and pop.



The functions associated with stack are:

- **empty()** – Returns whether the stack is empty – Time Complexity: O(1)
- **size()** – Returns the size of the stack – Time Complexity: O(1)
- **top() / peek()** – Returns a reference to the topmost element of the stack – Time Complexity: O(1)
- **push(a)** – Inserts the element 'a' at the top of the stack – Time Complexity: O(1)
- **pop()** – Deletes the topmost element of the stack – Time Complexity: O(1)

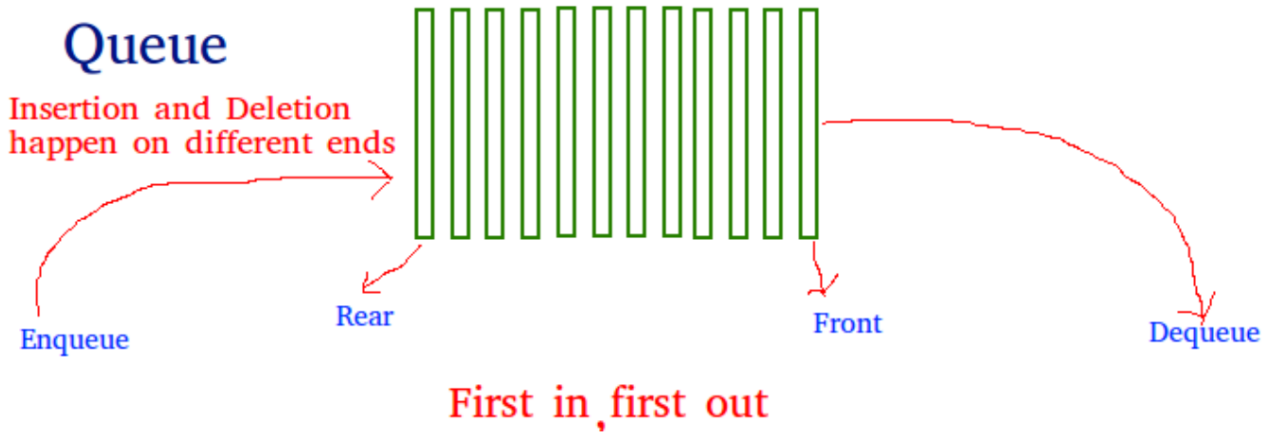
Queue in Python

https://www.geeksforgeeks.org/queue-in-python/

Queue in Python

Read Discuss Courses Practice

Like stack, queue is a linear data structure that stores items in First In First Out (FIFO) manner. With a queue the least recently added item is removed first. A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.

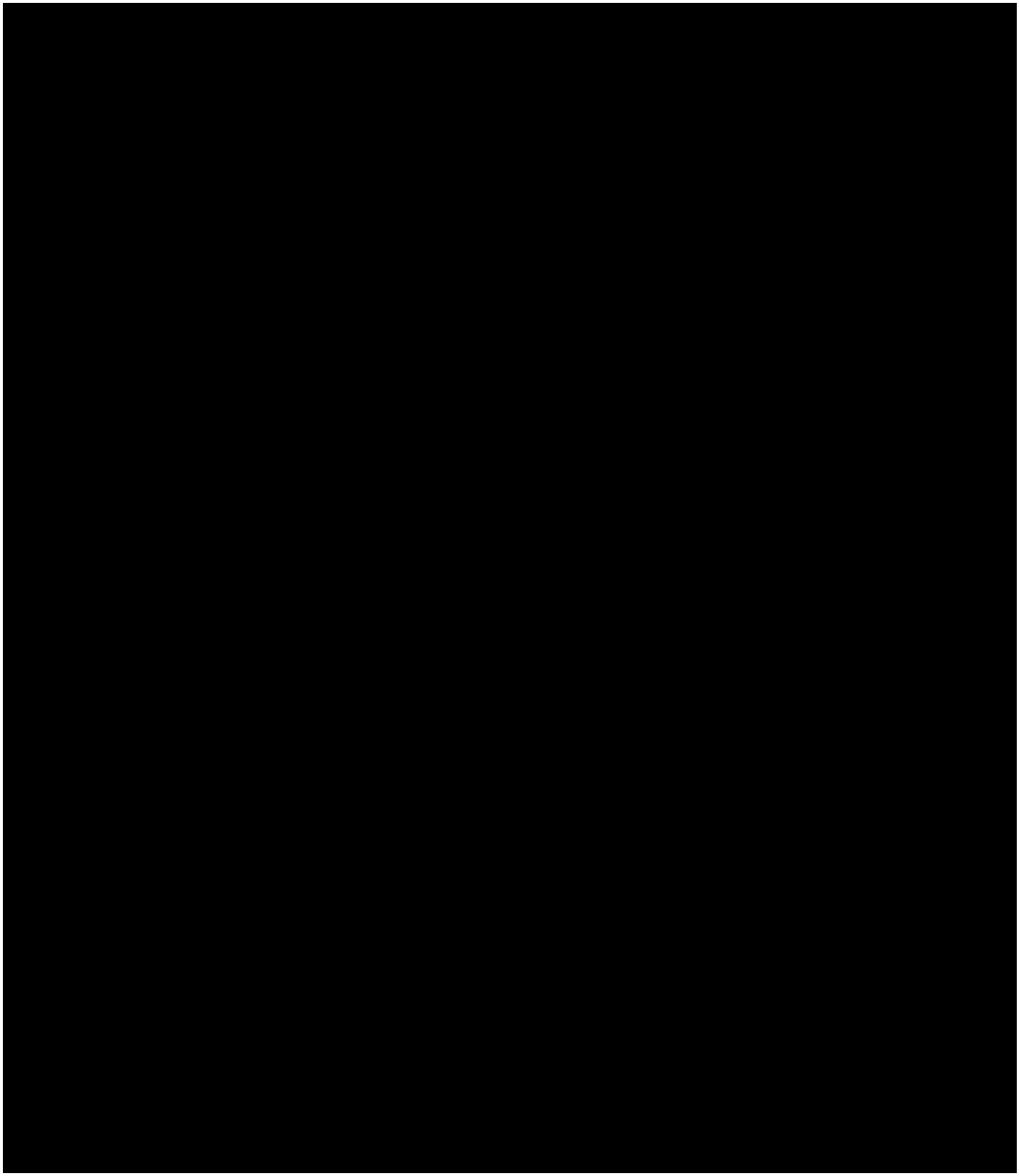


Operations associated with queue are:

- **Enqueue:** Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition – Time Complexity : O(1)
- **Dequeue:** Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition – Time Complexity : O(1)
- **Front:** Get the front item from queue – Time Complexity : O(1)
- **Rear:** Get the last item from queue – Time Complexity : O(1)

Heap queue (or heapq) in Python

https://www.geeksforgeeks.org/heap-queue-or-heapq-in-python/



5.4 程序生成测试数据

26971:分发糖果

greedy, http://cs101.openjudge.cn/routine/26971/

`n` 个孩子站成一排。给你一个整数数组 `ratings` 表示每个孩子的评分。

你需要按照以下要求，给这些孩子分发糖果：

- 每个孩子至少分配到 `1` 个糖果。
- 相邻两个孩子评分更高的孩子会获得更多的糖果。

请你给每个孩子分发糖果，计算并返回需要准备的 **最少糖果数目** 。

输入

第一行包含一个整数n。1 <= n <= 2 * 10^4
第二行包含n个整数，相邻整数间以空格隔开。0 <= ratings[i] <= 2 * 10^4

输出

一个整数

样例输入

```
Sample1 input:
3
1 0 2
Sample1 output:
5
```

样例输出

```
Sample2 input:
3
1 2 2
Sample2 output:
4
```

提示

tags: greedy

来源

LeetCode 135.分发糖果：https://leetcode.cn/problems/candy/

```python
import random
import time

random.seed(0)

for epoch in range(20):
    n = random.randint(10 + epoch * 2, 900 + 200 * (epoch // 2) ** 2)
    nums = [random.randint(0, 800 + 200 * (epoch // 2) ** 2 ) for _ in range(n)]
    inlines = [f'{n}\n'] + [' '.join([str(num) for num in nums]) + '\n']
```

```
10
11      with open(f'data/{epoch}.in', 'w') as f:
12          f.writelines(inlines)
13
14      def candy(ratings):
15          n = len(ratings)
16          left = [0] * n
17          for i in range(n):
18              if i > 0 and ratings[i] > ratings[i - 1]:
19                  left[i] = left[i - 1] + 1
20              else:
21                  left[i] = 1
22
23          right = ret = 0
24          for i in range(n - 1, -1, -1):
25              if i < n - 1 and ratings[i] > ratings[i + 1]:
26                  right += 1
27              else:
28                  right = 1
29              ret += max(left[i], right)
30
31          return ret
32
33      #input()
34      #*nums, = map(int, input().split())
35      start = time.time()
36      ans = candy(nums)
37      end = time.time() - start
38
39      print(f"[{epoch}] {end:.3f}sec")
40      print(ans)
41
42      with open(f'data/{epoch}.out', 'w') as f:
43          f.writelines([str(ans) + '\n'])
```

26976:摆动序列

greedy, http://cs101.openjudge.cn/routine/26976/

如果连续数字之间的差严格地在正数和负数之间交替，则数字序列称为 **摆动序列** 。第一个差（如果存在的话）可能是正数或负数。仅有一个元素或者含两个不等元素的序列也视作摆动序列。

- 例如，

[1, 7, 4, 9, 2, 5] 是一个 **摆动序列** ，因为差值 (6, -3, 5, -7, 3) 是正负交替出现的。

- 相反，

[1, 4, 7, 2, 5]

[1, 7, 4, 5, 5]

子序列 可以通过从原始序列中删除一些（也可以不删除）元素来获得，剩下的元素保持其原始顺序。

给你一个整数数组 nums ，返回 nums 中作为 **摆动序列** 的 **最长子序列的长度** 。

输入

第一行包含一个整数n。1 <= n <= 1000

第二行包含n个整数，相邻整数间以空格隔开。0 <= nums[i] <= 1000

输出

一个整数

样例输入

```
sample1 input:
6
1 7 4 9 2 5
sample1 output:
6

sample2 input:
10
1 17 5 10 13 15 10 5 16 8
sample2 output:
7
```

样例输出

```
sample3 input:
9
1 2 3 4 5 6 7 8 9
sample3 output:
2
```

提示

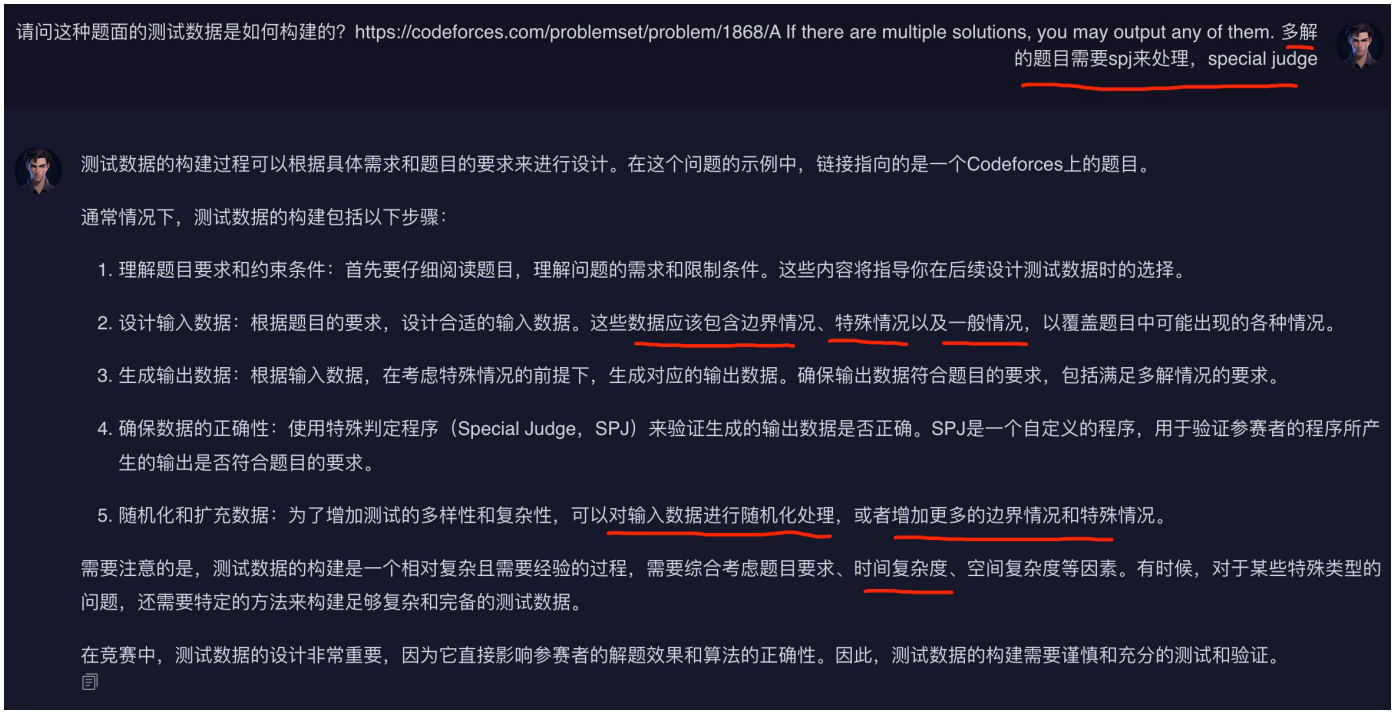
tags: greedy

来源

LeetCode 376. 摆动序列: https://leetcode.cn/problems/wiggle-subsequence/

```python
import random
import time

random.seed(0)

for epoch in range(20):
```

```
    n = random.randint(1, min(1000, 10 + epoch*50))
    nums = [random.randint(0, 1000 ) for _ in range(n)]
    inlines = [f'{n}\n'] + [' '.join([str(num) for num in nums]) + '\n']

    with open(f'data/{epoch}.in', 'w') as f:
        f.writelines(inlines)

    def wiggleMaxLength(nums):
        n = len(nums)
        if n < 2:
            return n

        prevdiff = nums[1] - nums[0]
        ret = (2 if prevdiff != 0 else 1)
        for i in range(2, n):
            diff = nums[i] - nums[i - 1]
            if (diff > 0 and prevdiff <= 0) or (diff < 0 and prevdiff >= 0):
                ret += 1
                prevdiff = diff

        return ret


    #input()
    #*nums, = map(int, input().split())
    start = time.time()
    ans = wiggleMaxLength(nums)
    end = time.time() - start

    print(f"[{epoch}] {end:.3f}sec")
    print(ans)

    with open(f'data/{epoch}.out', 'w') as f:
        f.writelines([str(ans) + '\n'])
```



CF1868A. Fill in the Matrix

tags: constructive algorithms （If there are multiple solutions, you may output any of them. 多解的题目需要spj来处理，special judge）

constructive algorithms, implementation, 1300, https://codeforces.com/problemset/problem/1868/A

There is an empty matrix M of size $n \times m$.

Zhongkao examination is over, and Daniel would like to do some puzzle games. He is going to fill in the matrix using permutations of length m. That is, each row of M must be a permutation of length m†.

Define the *value* of the i-th column in M as $vi = MEX(M_{1,i}, M_{2,i}, \ldots, M_{n,i})$‡. Since Daniel likes diversity, the *beauty* of M is $s = MEX(v_1, v_2, \cdots, v_m)$.

You have to help Daniel fill in the matrix M and **maximize** its beauty.

†† A permutation of length m is an array consisting of m distinct integers from 00 to $m-1$ in arbitrary order. For example, [1,2,0,4,3] is a permutation, but [0,1,1] is not a permutation (1 appears twice in the array), and [0,1,3] is also not a permutation ($m-1$=2 but there is 3 in the array).

‡‡ The MEXMEX of an array is the smallest non-negative integer that does not belong to the array. For example, MEX(2,2,1)=0 because 0 does not belong to the array, and MEX(0,3,1,2)=4 because 0, 1, 2 and 3 appear in the array, but 4 does not.

Input

The first line of input contains a single integer t ($1 \le t \le 1000$) — the number of test cases. The description of test cases follows.

The only line of each test case contains two integers n and m ($1 \le n,m \le 2 \cdot 10^5$) — the size of the matrix.

It is guaranteed that the sum of $n \cdot m$ over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, in the first line output a single integer — the maximum beauty of M.

Then output the matrix M of size $n \times m$ — the matrix you find.

If there are multiple solutions, you may output any of them.

Example

input

```
4
4 3
1 16
6 6
2 1
```

output

```
3
1 0 2
0 2 1
1 0 2
0 2 1
2
14 7 15 4 10 0 8 6 1 2 3 5 9 11 12 13
6
3 0 1 4 2 5
5 2 1 0 4 3
1 3 2 4 5 0
4 1 3 2 5 0
4 2 5 3 0 1
2 4 0 5 1 3
0
0
0
```

Note

In the first test case:

- $v1$=MEX(1,0,1,0)=2;
- $v2$=MEX(0,2,0,2)=1;
- $v3$=MEX(2,1,2,1)=0.

Therefore, s=MEX(2,1,0)=3.

It can be shown that 33 is the maximum possible beauty of M.

In the second test case, any permutation will make s=2.

In the third test case:

- $v1$=MEX(3,5,1,4,4,2)=0;
- $v2$=MEX(0,2,3,1,2,4)=5;
- $v3$=MEX(1,1,2,3,5,0)=4;
- $v4$=MEX(4,0,4,2,3,5)=1;
- $v5$=MEX(2,4,5,5,0,1)=3;
- $v6$=MEX(5,3,0,0,1,3)=2.

Therefore, s=MEX(0,5,4,1,3,2)=6.

5.4 testing_code.py

https://github.com/GMyhf/2024fall-cs101/blob/main/code/testing_code.py

```python
# ZHANG Yuxuan
import subprocess
import difflib
import os
import sys

def test_code(script_path, infile, outfile):
    command = ["python", script_path]  # 使用Python解释器运行脚本
    with open(infile, 'r') as fin, open(outfile, 'r') as fout:
        expected_output = fout.read().strip()
        # 启动一个新的子进程来运行指定的命令
        process = subprocess.Popen(command, stdin=fin, stdout=subprocess.PIPE)
        actual_output, _ = process.communicate()
        if actual_output.decode().strip() == expected_output:
            return True
        else:
            print(f"Output differs for {infile}:")
            diff = difflib.unified_diff(
                expected_output.splitlines(),
                actual_output.decode().splitlines(),
                fromfile='Expected', tofile='Actual', lineterm=''
            )
            print('\n'.join(diff))
            return False


if __name__ == "__main__":
    # 检查命令行参数的数量
    if len(sys.argv) != 2:
        print("Usage: python testing_code.py <filename>")
        sys.exit(1)

    # 获取文件名
```

```
    script_path = sys.argv[1]

    #script_path = "class.py"  # 你的Python脚本路径
    #test_cases = ["d.in"]  # 输入文件列表
    #expected_outputs = ["d.out"]  # 预期输出文件列表
    # 获取当前目录下的所有文件
    files = os.listdir('.')

    # 筛选出 .in 和 .out 文件
    test_cases = [f for f in files if f.endswith('.in')]
    test_cases = sorted(test_cases, key=lambda x: int(x.split('.')[0]))
    #print(test_cases)
    expected_outputs = [f for f in files if f.endswith('.out')]
    expected_outputs = sorted(expected_outputs, key=lambda x: int(x.split('.')[0]))
    #print(expected_outputs)

    for infile, outfile in zip(test_cases, expected_outputs):
        if not test_code(script_path, infile, outfile):
            break

```